Insights from the HARP Framework: Using an AI-Driven Approach for Efficient Resource Allocation in HPC Scientific Workflows

Swathi Vallabhajoyula*
vallabhajosyula.2@buckeyelink.osu.edu
The Ohio State University
Columbus, Ohio, USA

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

ABSTRACT

High-performance computing (HPC) is essential for scientific research, and efficient utilization of such high-demand resources requires end users to understand their scientific workflows or tools and their synergy with the execution environment. There needs to be more workflow-specific history for machine learning (ML) models to estimate resource requirements tailored to a specific workflow or set of applications. In this work, we present the potential problems encountered while manually finetuning a workflow for optimal resource utilization without overprovisioning or underallocating the resources. We highlight the need for an AI-driven framework to estimate the resource requirements of applications within the workflow and recommend optimal resource allocation configurations. We introduce our generalizable AI-driven application, the HPC Application Resource (runtime) Predictor (HARP) Framework. HARP generates execution history against application parameters and available hardware, builds several regression models, and selects the best model to recommend cost-based or time-based configurations for optimal resource allocation. Our work demonstrates the effectiveness of HARP for estimating resource requirements against Ohio Supercomputer Center (OSC) for training a fully-connected neural net for image classification.

CCS CONCEPTS

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

KEYWORDS

automated data generation, ML, model scalability, model transferability, walltime estimation,

ACM Reference Format:
Swathi Vallabhajoyula 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 (PEARC '23), July 23–27, 2023, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3569951.3597595

*Both authors contributed equally to this research.

PEARC '23, July 23–27, 2023, Portland, OR, USA

ACM ISBN 978-1-4503-9985-2/23/07…$15.00
https://doi.org/10.1145/3569951.3597595

1 INTRODUCTION AND BACKGROUND

Various academic and commercial supercomputing centers, such as Ohio Supercomputer Center (OSU), Texas Advanced Computing Center (TACC), San Diego Supercomputer Center (SDSC), and Amazon Web Services (AWS), offer different types of services, including batch allocations, web servers, and dedicated nodes to perform HPC like weather prediction, or batch submissions for gene sequencing, or training large Deep Neural Network Models. The cost of each request is determined based on several factors, such as the type of service, hardware, allocation time, and provider policies. Based on prior knowledge about the computing needs of their workflows, users select configurations that meet the necessary cost/time limits.

1.1 Problem Statement:

A job is said to be efficient if it utilizes the allocated resources (cores) to their fullest extent by parallelizing the code if requesting multiple cores, overlapping compute-I/O cycles, and efficient multithreading (inter-intra op thread distribution). For instance, if a job requests a full CPU node (28 cores per node), OSC allocates a complete node with 28 CPU cores and complete node memory (≈117GB). In order to be considered efficient, the job must consume the entire memory block or use all 28 cores for most of the allocation time.

To best use shared cyberinfrastructure (CI) resources, users need to understand the requested node architecture, allocation times, and the space and time complexities of their workflow applications and tools. It is crucial to understand the type of application, whether it requires serial or parallel executions, is compute or I/O-intensive, and whether CPU or GPU-intensive, to make the appropriate node requests. By understanding the relationship between cores per node and memory allocations, GPUs per core, and GPU Memory [1], users can optimize the cores per node based on memory requirements or desired parallelism for faster executions or lower per-core hour billing.

Users who request and utilize shared computing infrastructure (CI) can be classified into three categories: beginner, intermediate, and expert, based on their experience and skills. Beginners typically submit job allocations with recommended defaults and may adjust the allocation hours if their jobs fail due to under-allocations. As they submit more jobs with different workflows, they gain some experience through application and HPC documentation. This group of users (now with intermediate experience) can write custom codes

[1] https://www.osc.edu/content/academic_fee_model_faq

to parallelize computations, data loading, and optimize I/O operations. Expert users have a deeper understanding of the underlying hardware and software, and they often develop their tools or codes to optimize the execution time of their jobs. To study user behaviors, we observed diverse research groups at OSU, including students performing bioinformatics, research groups implementing small to large AI-driven models for interdisciplinary sciences, and developing optimally distributed high-performance deep learning (HIDL) applications.

Users typically share budgeted projects or groups on shared CIs and are charged per node type and core hour for their utilization expenses. Most jobs are submitted with defaults and only adjusted if the job fails or drains most of project allocation. Users tend to be more mindful when requesting from more expensive CIs (like AWS). We observed that users request full-node (all cores) for GPU jobs as a default allocation leading to low CPU utilization. Optimizing resource allocation can significantly reduce the cost of executing a job. We found that up to **40%** of allocation costs could be saved if node allocations were adjusted based on memory requirements, GPU [1] and parallelism [2].

1.2 Contributions

An AI-driven framework can observe the executions (job traces on CIs) and estimate the resource requirements based on application hyperparameters, allocations, and execution history. The model should synthesize the runs when training data is unavailable and tailor the estimations with respect to a target application (job). We explored the feasibility of building a framework that can profile a given job and estimate the walltime based on the availability of resources (configurations). Generating training data for a given job and configuration is time-consuming and resource-intensive (especially when each execution can take anywhere from a few hours to days). For instance, training the DNN model with various hyperparameters for ImageNet data classification to achieve better accuracy (epochs=100) can be expensive. In our previous research[3], we investigated the possibility of generating cost-effective, application-specific training data by scaling down the executions to understand the distribution of execution patterns and adding a few complete (full) executions to scale the predictions. We also explored the training data re-usability with applications in the same family using the data obtained from training a VGG16 model to estimate the resource requirements for ResNet50. We developed various regression models and proposed a custom policy to choose the best model per target application. In our previous work [4] we proposed a preliminary framework that estimates runtime for a given application. In this work, we expanded on the framework and created an end-to-end framework named HARP, which predicts HPC Application Resources (runtime). The framework can predict the best execution configuration for a given target application based on time/cost constraints. HARP can be configured to generate training data, build regression models, and predict execution times for a given application against the provided configurations. The HARP (release 1.0) is ready to be installed on Linux-based systems and tested on OSC clusters. The code is available to download from GitHub [2].

[2]https://github.com/ICICLE-ai/harp

1.3 Challenges for building an AI-driven framework that estimates resource requirements for custom workflows.

Several factors influence the execution time and resource needs making it difficult to estimate resource requirements easily. In paper [3], we observed that each application has a unique set of configurations that influence its behavior (memory needs and execution times), and a change in one (or more) can change the resource requirements drastically. Users estimate their resource requirements based on prior runs and often re-execute the job with the same allocations. Configuring their job with different parameters (say, training DNN with large epochs or a smaller/larger batch) calls for a change in allocations. Increasing batch size increases memory requirements, thus pushing the request to a larger GPU node which is expensive and often has long wait times. Pushing it to a CPU makes it run slower. In scenarios with diverse correlated and non-linearly related features influencing the execution time, machine learning models are needed to identify patterns and estimate the resource needs. They can predict known execution resources and can also scale to unseen environments and hyperparameters. While these models have potential, there are challenges in building and using these models. We detail these challenges next.

1.3.1 Availability of execution history: Good history of executions is needed to train models to learn utilization patterns (memory requirements, execution times) w.r.t allocations (hardware - memory, CPU/GPUs requested), and application input (DNN training data) and hyperparameters (optimization, batch size). Supercomputing centers are often constrained from sharing their workload due to considerations of competition and client confidentiality. Some datasets are indeed available, such as the MIT Data Center challenge [3] dataset created to profile and estimate workloads of DNN applications. This dataset has high-level information about the job type (DNN model and ID) and their respective execution allocations. The details about the execution itself (batch size, input training data, optimizations) are provided as low-level time-series event logs(≈5TB). Due to the size and need for additional models to extract more information about the execution, we decided to focus on generating target application-specific execution traces efficiently (quick and less expensive).

1.3.2 Understanding and capturing resource-specific features for machine learning: Depending on the nature of the task at hand, an application can be classified as compute-intensive (such as training neural networks), I/O intensive (such as processing large text corpora), and/or memory-intensive (for instance, training Deep Neural Network models like BERT [5]). These workloads possess distinct features that describe their characteristics, including application hyperparameters and hardware attributes like CPU/GPU frequency, cores, memory, bandwidth, cores per node, and threads spawned. Effectively engineering these features requires an in-depth comprehension of the tool or application's implementation, resource allocation, and predictive model characteristics. As a result, customized feature engineering for each workflow and hardware architecture can be challenging, time-consuming, and unproductive. To overcome this challenge, we need a framework that can automatically

[3]https://dcc.mit.edu/data

capture such features from execution history using human-in-loop to provide different execution configurations specific to the domain/job.

1.3.3 Choosing an appropriate regression model per application-specific data: To select an appropriate machine learning (ML) algorithm, data scientists consider factors such as the type of training data and the availability of training samples. For instance, some models work well if the relationship between input features and output predictions is linear (such as multi-linear regression or MLR), while others only work with numeric data (also MLR or neural network regressors, NNR) or combinational data (such as Decision Trees(DTR)). Some algorithms require a large number of training samples distributed across features (such as NNR), whereas others only require a few samples (such as DTR). Moreover, as the target application changes, the characteristics of the features may also change, and selecting a universal ML algorithm for making estimations may not result in the best predictions. Thus, we require an algorithm to choose the best regression models ad-hoc for each targeted application.

1.3.4 Selecting the appropriate execution configuration: It is possible for different allocation configurations for a job, all of which execute the job more-or-less efficiently. Some configurations may result in faster execution times, while others may have lower cost of execution. The researcher is now stuck with identifying which configuration is best. In other words, it should be the responsibility of the *framework* to select the most appropriate prediction model and execution configuration based on the time and cost constraints of the job.

2 ADDRESSING CHALLENGES - THE HARP FRAMEWORK

In this section, we describe the three components - "generate", "build" and "predict" - that form the heart of HAPR.

(1) **The human-in-loop "generate" component:** Using the CODAR Framework [4], this component generates training data by running applications against the provided scaled-down (SD) and full-scale (FS) configurations. The application-specific hyperparameters are defined by the HARP user (i.e. the end-user). In contrast, the HARP developer only needs to define the execution endpoint configurations to generate training data under "scaled-down" (SD) and "full-scale" (FS) executions. The SDs are used as training data, and gradually, FS data is incrementally included to validate and test the framework with "seen/unseen feature values." The SDs are generated by executing the application at a smaller scale, creating less intensive/faster workloads that enable executing them with a different and diverse set of potential applications or allocation configurations quickly and efficiently. These executions help models to understand the synergy between these variable features. The FS runs are configured to project this information to a different scale (of input or hardware).
(2) **The automated "build" component:** The framework captures all possible features that directly or indirectly contribute towards estimating walltime. They could be unknown contributors (like memory, which changes the cores per node allocation, which in turn changes the execution time) or presumed features (as known features), which could be redundant (omp-num-threads and cores allocated per node). This component pre-processes the generated training data, removes redundant features, and combines correlated features to reduce dimensionality and improve the accuracy of predictive models. We train three-different regression models (MLR, DTR, Simple NNR) against different combinations of training data (SD, SD+n%FS) and store the models for inference. We keep training the models with incremental FS sample generation (%FS) to replicate the human learning. We up-sample FS data to remove the SD-FS imbalances and scale the predictions w.r.t FS validation dataset (adjustment factor).
(3) **The policy-driven "predict" component:** The final component of the framework predicts the execution time for the inference(test). The best-performing model is selected using the mean-absolute-error (MAE) metric or a custom "under-prediction and over-estimation" (Walltime Metric - WTM) policy built combining several metrics (like under-prediction percentage (UPP), over-estimation mean absolute percentage error (OV_MAPE)) [4]. A model performs better when it has lower UPP and minimizes the OE_MAPE. The framework chooses an optimal predictive model (OPM) based on the policy and estimates the walltime against all the available configurations (hardware & application). The optimal allocation configuration is selected based on time/cost limitations.

[4] https://github.com/CODARcode/cheetah

3 A USE CASE FOR HARP - ESTIMATING RESOURCE REQUIREMENTS FOR A TARGET APPLICATION

This section provides an example that illustrates the capabilities of HARP. Our objective is to evaluate the performance of a fully-connected neural network image classifier by estimating the runtimes for a pre-defined set of configurations. Prior to executing HARP, we provide the SD and FS configurations by varying the input datasets (mnist, cifar10) and sample size, hidden layers, batch sizes, and other Deep neurl network (DNN)-specific hyperparameters. It generates training, processes it (as shown in Figure 1) and estimates walltime for each test configuration. The framework generates and pre-processes the training data based on the provided configurations and selects the best application-specific regression model to "predict" execution time for configurations meeting cost/time limitations. The framework reduced the cost of generating "history" by a fifth after scaling-down executions (SD). (It took ≈3.5 Hrs to generate SD and FS training data as opposed to ≈19.4 Hrs if they were generated at full-scale.) The framework is evaluated for predicting DNN training walltime against test data with seen distributions and unseen distributions like GPU node(a100), input data dimensions, higher dimensionality for neurons per layer, hidden layers, batch, and epoch sizes.

The "predict" phase has the following steps:

(1) A "basic" vs. "custom" prediction - Predicting walltime as one component (basic) vs. predicting walltime in terms of

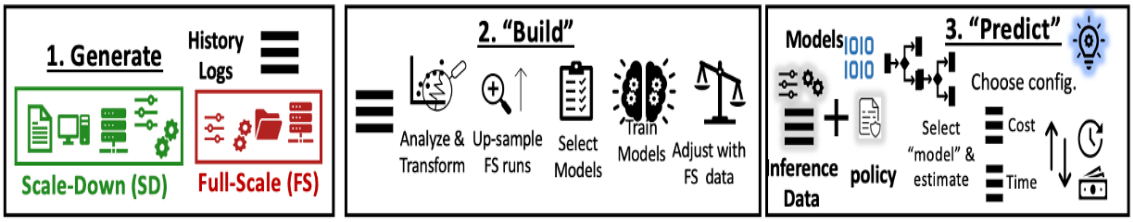

Figure 1: Visualizing the Automated Data Collection Pipeline and Model Generation

Est.	Basic		Custom		Deft.
Policy	**WTM**	**MAE**	**WTM**	**MAE**	
(RM, TD, VA)	*(DTR, SD, 1.29)*	*(NN, SD+FS, 1.04)*	*(NN, SD, 1.55)*	*(NN, SD, 1.55)*	*N/A*
MAE	188	145	**46**	**46**	-
MAPE	105	102	**74**	**74**	-
UPP	58	62	**71**	**71**	-
OV_MAPE	139	166	**80**	**80**	-
WT Co.	448	430	**439**	**439**	**207**
$ Co.	0.02	0.02	**0.02**	**0.02**	**33.22**

Config	(48, 1, 48)	(28, 1, 28)	(10, 1, 10)
Act. WT	685	516	516
Act. $	0.045	0.025	0.017
Dflt. $	160.351	89.71	61.887
H Pred.*	180	*95*	*103*
	0.012	*0.005*	*0.003*
H Exec.*	1225	*1180*	*1237*
	0.08	*0.57*	*0.041*
Cost Lt.			X
Time Lt.		X	
* - WT (Walltime) and $ (execution cost);			

Table 1: (Left) The table shows the Walltime predictions made by HARP in "Basic" and "Custom" mode against selected policy (regression model (RM), data (TD), adjustment (VA)). The default (Deft) cost is calculated for allocating full nodes with a max walltime limit. (Right) Applying OMP to select the best configuration w.r.t. cost (e.g. $0.01) and time (e.g. 120 sec) limits on H.Pred. H-Exec shows the time (cost) spent on executing the job(config.) with H-Perd. Wt. and $ Co. gives the average per model H-Exec values

sub-components (custom: epochs * per-epoch + constant time to load libraries and datasets to memory). From Table 1 (left), Predictions made with a custom estimator have lower MAPE and OV_MAPE errors. Customizing the estimators by inducing domain knowledge helps reduce prediction errors.

(2) Policy-based regression model selection - Selects best-performing regression model per application based on Policy. From Table 1. (left), we can see the change in selection policy changing the prediction model (RM, TD, VA). The WTM policy emphasizes reducing the underpredictions and minimizing the over-estimation error.

(3) Cost/time bound execution configuration selection - Given a hard limit on time (or cost), select the configuration with minimum cost (or time). We strike the cost-time balance by selecting a less expensive configuration that meets the time limit (or vice-versa). Table 1 (right) shows an example where the framework selects (28, 1, 28) as a configuration whose predictions meet the time limit and has a lower cost than the other qualifying peers.

4 CONCLUSION AND FUTURE WORK

We tested the feasibility of the pipeline against three unique single-node workloads[4]. We were able to induce information about core-hour billings and OSC's CI architecture into the framework. This enabled us to estimate predictions as per OSC allocations and budgets. We explored the feasibility of model transferability between DNN models by encoding model type or describing a network (no. of hidden layers, neurons, opt). We propose future work in three areas: (a) Adding more CIs into the framework and (b) Profiling distributed training or workloads. and (c) building model commons to re-use models to estimate allocations for repetitive or similar workloads.

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.*

REFERENCES

[1] Baolin Li, Viiay Gadepally, Siddharth Samsi, and Devesh Tiwari. 2022. Characterizing multi-instance gpu for machine learning workloads. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 724–731.

[2] Achille Peternier, Walter Binder, Akira Yokokawa, and Lydia Chen. 2013. Parallelism Profiling and Wall-Time Prediction for Multi-Threaded Applications. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering* (Prague, Czech Republic) *(ICPE '13)*. Association for Computing Machinery, New York, NY, USA, 211–216. https://doi.org/10.1145/2479871.2479901

[3] 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

[4] Swathi Vallabhajosyula and Rajiv Ramnath. 2022. Establishing a Generalizable Framework for Generating Cost-Aware Training Data and Building Unique Context-Aware Walltime Prediction Regression Models. In *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 497–506. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00070

[5] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962* (2019).