2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10826096

Toward Smart Scheduling in Tapis

Joe Stubbs
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
jstubbs@tacc.utexas.edu

Smruti Padhy
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
spadhy@tacc.utexas.edu

Richard Cardone
Texas Advanced Computing Center
University of Texas at Austin
Austin, TX, USA
racardone@tacc.utexas.edu

***Abstract*—The Tapis framework and APIs enable a wide range of research and scientific discovery by automating job execution on remote resources, including HPC clusters and servers running in the cloud. While Tapis simplifies interactions with cyberinfrastructure (CI), users still need to specify all details of a job they want to run, including the system, queue, node count, and maximum run time, among other attributes. Moreover, system and application resources must be defined in Tapis before a job is submitted. It can be a challenge for scientists to provide exact configurations and resource requirements for their jobs, which if incorrectly specified can delay their work.**

In this paper, we investigate developing an intelligent job scheduling capability in Tapis, where job configuration can be partially automated and computational resources can be dynamically provisioned by Tapis. We outline an architecture for such a feature and identify a set of core challenges to be solved. Then, we focus on one specific challenge: predicting queue times for a job on different HPC systems and queues, and we present two sets of results based on machine learning methods. Our first set casts the problem as a regression, which can be used to select the best system from a list of existing options. Our second set frames the problem as a classification, which allows us to compare the use of an existing system with a dynamically provisioned resource.

***Index Terms*—Tapis, HPC Jobs History, Smart Scheduling, Histogram-based Gradient Boosting Methods**

1. Introduction

Tapis [1] is a cloud-hosted API framework for reproducible computational research with thousands of active users across various research domains. A primary feature of Tapis is the ability to execute jobs on remote systems on behalf of users, including batch jobs on HPC clusters and high-throughput jobs on cloud servers. While an individual user may have access to several systems – from multiple HPC clusters at different centers, each with different queues, to cloud and high-throughput servers running on campus clusters or public clouds – Tapis currently requires the user to specify the exact parameters for each job, including the system to run on, the queue, if applicable, and other aspects, such as the number of nodes to use, the maximum runtime for the job, etc. Moreover, the system resource to be used for the job must be configured in Tapis prior to submitting the job, precluding solutions where resources are provisioned automatically and "just in time" for a specific user's workload. While in some cases users may wish to specify exact details about where and how their job should run, in other cases, the user's primary objective is simply to get the analysis performed as quickly as possible. Additionally, they may lack detailed knowledge about the characteristics of different systems, queues and/or the application, making it difficult or impossible for them to provide an optimal job configuration.

In this paper, we present our work to date to develop a smart scheduling job capability in the Tapis framework. We define the *smart scheduling with dynamic resource provisioning* problem as follows: given a user-supplied job submission request with partial configuration, automatically determine the complete job configuration which optimizes some objective function, considering all systems/queues available to the user and, if applicable, the possibility of dynamically provisioning dedicated resources for the job. In general, the objective function can be thought of as a cost function to be minimized, where cost could be measured in different ways, e.g., time, service units, dollars, CO2 emissions, etc. In this work, we focus on the objective of minimizing *time-to-solution*, that is, the total time from when the user submits the job to when the results of the job are available.

Developing such a feature presents a number of challenges which we describe in detail in Section 3, including automatically determining the following: 1) *system attributes*, including hardware architecture and software dependencies; 2) *job constraints*, including required hardware resources (CPU cores, memory) and total runtime; 3) *data movement cost*, that is, the time required to stage input data into and archive resulting output from the execution host, and 4) *system provisioning and queue time*, that is, the total time to provision a resource or wait for a job (in queue) to start running. We believe these challenges represent core problems to be tackled by the research CI community, the solutions to which would enable improved usability and utilization of the underlying cyberinfrastructure.

In the remaining sections of the paper, we focus on challenge 4) and machine learning (ML) methods for estimating queue time for batch-scheduled HPC systems. Using historical data from the Stampede2 system at the Texas Advanced Computing Center (TACC), we develop two sets of ML models for predicting queue time. In the first case, we develop regression models that predict the real-valued queue time for a job based on a set of six attributes. Two attributes (*num_nodes* and *max_minutes*) are part of the job resource request while the remaining four (*backlog_minutes*, *backlog_num_jobs*, *running_num_jobs*, and *running_minutes*) capture current queue state at the time of job submission. These predictions can be used to compare queue time across a set of existing systems and queues. In the second set of results, we use the same six attributes to develop classification models that predict the *queue time bin*, that is, the range of minutes that the job's queue time will likely fall into. The predicted queue time bins organize jobs according to the model's confidence in their candidacy for dynamic provisioning, where jobs classified in the highest bins represent candidates for which the system has the greatest confidence that the dynamic provisioning cost will be considered worth paying.

We utilize a software framework for automatically searching across a configurable space of model types and hyperparameters. Our best classification and regression models use Histogram-based Gradient Boosting and achieve over 90% accuracy (respectively, over 0.9 R^2 score) on held-out test sets across a six month range of time. These and related results are presented in Section 6.

2. Background

2.1. Tapis

Tapis [1] is a cloud-based infrastructure for advanced computing tasks developed by the University of Texas, Austin. Using Tapis, researchers manage petabytes of different types of data across file servers and object stores, execute scientific research codes on high-throughput and high-performance computing clusters, store metadata about their projects, share results with their colleagues, and publish their work to the greater research community.

Tapis has been used by thousands of researchers across more than 30 production tenants and projects funded by DARPA, NASA JPL, NIH and NSF. Tapis has enabled research across a wide range of fields – from agriculture and animal ecology, to civil engineering, climatology, and epidemiology, to neuroscience and synthetic biology – and studying some of the most significant challenges facing society, including combating climate change, fighting global pandemics and searching for exoplanets. The Tapis API serves hundreds of millions of requests and moves multiple petabytes of data each year.

2.2. ML Metrics

We used three standard metrics for all the regression techniques to evaluate the models: R^2 score (r2Score), mean absolute error (MAE), and root mean square error (RMSE). R^2 [1] provides measure of goodness of fit, that is, how well the independent variables explain the variance in the dependent variables and predict unseen data samples output. It is defined as follows:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\hat{y}$ is the predicted value. R^2 score ranges from 0.0 to 1.0, the best model score being 1.0 and the constant model score is 0.0. The value becomes negative when the model is the worst.

MAE[2] measures the mean absolute error which is the expected value of the absolute error. It is defined as follows:

$$MAE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i|$$

RMSE[3]measures the square root of the mean squared error. It is defined as:

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2} \quad (1)$$

3. Smart Scheduling and Associated Challenges

Our goal is to minimize *time-to-solution* by predicting queue time. Let *time-to-solution* = *initiation_time* + *data_move_time* + *execution_time*, where *initiation_time* is either the time spent in queue or the time spent provisioning infrastructure. This paper examines only the *initiation_time* component of *time-to-solution*.

In drilling down into the problem of dynamically scheduling batch jobs, we focus on Tapis's design but the challenges are broadly applicable. Our first task is to select the set of candidate hosts on which a job can run. Each system is characterized by its hardware and software attributes, and each application is constrained to run only on systems with the attributes it requires. For example, an application may be constrained to run only on x86-64 processors, so hosts with ARM processors are excluded. Applications and systems must be compatible: Systems advertise their attributes, and applications are matched to systems that meet their constraints.

1. https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score
2. https://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error
3. https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error

3.1. System Attributes and Job Constraints

Standards such as Redfish [2] and SNMP [3] provide extensive vocabularies of system traits. These vocabularies fully characterize a host's installed hardware, firmware and software. However, they also introduce a level of complexity and precision not necessary at the application level.

In 2021, SGCI published a resource description specification [4], [5] to improve interoperability between data centers and, specifically, to allow applications to select hosts on-the-fly based on compatibility and available capacity. Integration with several frameworks including Tapis was prototyped, but promoting a new standard proved to be resource intensive.

Tapis's current design direction is to use dynamic sets of system attributes that do not have to be predefined. System definitions can be annotated with key/value pairs to advertise their capabilities. Applications specify boolean expressions that reference system attributes and are evaluated at runtime. When an expression returns true, the system meets the application's constraints and becomes a candidate for execution. This dynamic approach is flexible, easy to implement and requires only local agreement on vocabulary among applications and the systems they use.

3.2. System provision and queue time

While high-throughput systems such as cloud servers typically start user workloads immediately, most HPC systems utilize a batch scheduler where submitted jobs wait in queue to execute. The wait times vary and depend on factors such as the current state of the queue as well as characteristics of the job. The queue time can represent a significant portion of the overall time to solution; some jobs at TACC can wait in queue for more than 24 hours before starting. Thus, estimating the queue time for a specific job and resource represents an important aspect of computing the time-to-solution objective function.

Dynamically provisioning a resource for a specific job broadly involves instantiating virtual servers with the storage, networking and software required for the application to run correctly using an API such as AWS EC2 or JetStream2's OpenStack API. Various methods exist for minimizing the total time required for resource provisioning. In this paper, we assume a fixed (i.e., constant) provisioning time to dynamically deploy a job-specific resource. Of course, dynamically provisioning a resource still consumes physical resources from some system and has the potential for other impacts, such as incurring a real cost (in dollars) when using resources on a public cloud. In practice, the consideration to use a dynamically provisioned resource for a job could be a complex decision depending on various factors, such as time-sensitivity of the computation and the availability of other resources. We formalize this trade-off by introducing a *tolerance factor*, a positive real value quantifying the extent to which using existing systems is preferred over dynamically provisioning. To be precise, let q denote the time a job waits in queue on an existing system, let p denote the time to dynamically provision a resource for a job, and let t be the tolerance factor. Then dynamically provisioning a job is desirable whenever $q > p * t$, while $q \leq p*t$ implies that waiting in queue on the existing system is desirable.

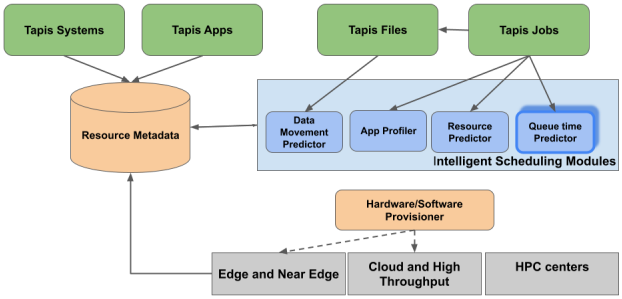

Figure 1. Architecture for Tapis Smart Scheduling

In the rest of the paper, we focus on predicting queue times for jobs on existing clusters with the goal of comparing candidate systems to each other as well as to a dynamically provisioned resource. For comparing existing systems to each other, we frame the problem as a regression where the system with the smallest predicted queue time would be selected. For comparing existing systems to a dynamically provisioned resource, we analyse it as a classification problem, where the goal is to predict the queue time bin of a job on a system. Queue time bins are continuous ranges of minutes, and we study the classification problem for different numbers of bins of size $p * t$. In this setting, jobs predicted to be in bin two or higher are candidates for dynamic provisioning, with jobs in the highest bin being the candidates predicted to benefit the most.

4. Architecture

Smart scheduling in Tapis will integrate a number of new modules into existing Tapis components. The high-level architecture is depicted in Figure 1. Metadata about available compute resources is maintained in a new Tapis knowledgebase and updated on regular intervals (e.g., every few hours). This information includes "static" metadata about hardware architecture, resources (e.g., memory, CPU cores, disk, number of nodes), as well as "dynamic" information, such as number of jobs in queue (batch clusters) or battery life (IoT devices).

A set of new "Intelligent Scheduling Modules" profile the applications to be scheduled and provide cost estimates for different aspects of the scheduling lifecycle. The App Profiler module determines hardware requirements, such as compatible architectures, and extrapolates runtime requirements, such as memory, CPU cores, and runtime, for an application based on sampling of inputs. By combining metadata about available computing resources with an application's profile information, the Tapis Systems and

Apps services produce an initial list of potential execution targets will be produced. Each of these execution targets will then be evaluated to determine which optimizes the overall scheduling objective function.

The Data Movement Predictor, Resource Predictor, and Queue Time Predictor modules estimate the objective function for the corresponding tasks. Note that these are job-specific, as details about the job, including the file inputs and parameters, will impact the estimates. The Data Movement Predictor will be invoked by the Tapis Files service, since it is aware of details regarding possible optimizations that can be performed when transferring between systems (e.g., compression). The other predictors will be invoked by the Jobs service which will aggregate all of the predictions and ultimately make a scheduling decision.

5. Methods

5.1. Data Sources and Preprocessing

The Texas Advanced Computing Center (TACC) uses tacc_stats [6] and Slurm to record information about every job submitted to any of its HPC systems, which gives us queue time in minutes for each job. We focused our attention on 6 months of cleaned, historical data for Stampede2. Specifically, we worked with 2022 data (Feb 1 to Jul 31) for two production queues, *skx-normal* and *normal*, which schedule jobs on SKX and KNL nodes, respectively.

5.2. Exploring different techniques

5.2.1. General approach. We performed standard exploratory data analysis, visualization, and feature selection using the Python `pandas` ([7], [8]) and `matplotlib` [9] packages. We removed features such as *jobid*, *user*, *start_time* and *end_time* from the training, test, and validation sets because they were either non-predictive in early results or required future knowledge. We selected the features *num_nodes* and *max_minutes*. We then engineered features that reconstruct system state *at the time each job was submitted*. Specifically, we derived for each job *backlog_num_jobs* and *backlog_minutes* to record the number of jobs in queue and their total requested minutes, respectively. Similarly, *running_num_jobs* and *running_minutes* represent the number of running jobs and their total requested minutes. We applied standard techniques such as data shuffling and handled outliers. Any job whose waiting time was greater than two days was considered an outlier, given the maximum duration for a job request in Stampede2 is two days. This resulted in the removal of 5831 jobs (2.26%) from the *skx-normal* queue dataset (169114 jobs) and 1821 jobs (1.07%) from the *normal* queue dataset (257053 jobs).

We developed a model search and evaluation framework in Python [10] that utilizes a configuration file to explore various model types and associated hyperparameter spaces, as well as other configurations such as the dataset to use. For a given input dataset, we developed a configurable sliding window method to split sub-intervals of the data into a *current* and *future* set. The *current* dataset was further divided into training and test sets using an 80/20 split. For example, given an initial data set with six months of jobs data, we could create six one-month windows and split each window into a *current* and *future* set. We experimented with window splitting based on time as well as job counts. Models would then be trained on the training subset of the *current* set and evaluated on the test subset of *current* as well as the *future* set. All of these settings can be assigned in the configuration file, allowing the program to run for days uninterrupted.

5.2.2. Regression techniques. Queue time prediction can be modeled as a regression problem. Several regression techniques have been used in the literature based on different use cases and data([11]–[14]). We modeled the problem both as a time-series and non-time-series regression. For non-time-series models, we selected the six features listed in Section 5.2.1 and applied the following machine learning techniques: Linear Regression, K-Nearest Neighbor (kNN), and Histogram-based Gradient Boosting (HGB). For time-series models, we added extra features commonly used in time-series data preparation. We then applied Linear Regression, Lasso, Feed-forward Neural Network, and Long Short-Term Memory by partitioning the data and sliding the window to train and test the data. In the end, we observed that the results were worse than in the non-time series cases.

5.2.3. Classification techniques. We also modeled queue time as a classification problem, where the goal is to predict the correct queue time bin for each job. We define queue time bins as continuous ranges of minutes of a fixed size, except for the last bin, which has no upper bound. We explored different sizes and number of bins. For example, in the case of 4 bins of size 60 minutes, the goal is to predict whether the job's queue time will be: 0-60 minutes, 60-120 minutes, 120-180 minutes or greater than 180 minutes. As in the case of regression, we explored various models and associated hyper-parameter spaces, including Histogram-based Gradient Boosting (HGB), K-Nearest Neighbor (kNN), Logistic Regression, Random Forrest (RF), and Support Vector Machines. We also used the sliding window technique for training and testing, using accuracy against known queue times as our validation metric.

6. Evaluation and Results

We settled on a 90/10 *current/future* split in each of six 1 month windows as described in the Section 5.2.1, and further partitioned *current* into an 80/20 *training/test* split. We modeled using both regression and classification with the features described in Section 5.2.1. We tried the seven regression and five classification techniques listed in Sections 5.2.2 and 5.2.3, respectively. The repository [10] contains the raw data, the processed data, the python model search framework and the results.

Table 1 shows the kNN and HGB outcomes, which produced our best results. We highlight only HBG results here because of its computational advantage. We used three metrics for all regression techniques to evaluate the models: R^2 score (r2Score), mean absolute error (MAE), and root mean square error (RMSE), the latter two in minutes. We ran a model search framework with different configurations for different model types. For the HGB regressor model from scikit-learn [15], we ran the search with learning rates ranging [0.01, 0.1, 1.0], maximum trees ranging [10, 100, 400, 500, 600] and maximum depth ranging [3, 5, 9]. The parameters maximum trees = 500, maximum tree depth = 9, and learning rate = 0.1 for *skx-normal*, it yielded a high accuracy of r2_score=0.9 and low MAE=57.85 and RMSE=147.72. These predictions help choose a system and queue to run a job dynamically.

We ran HGB Classifier to predict the bin into which a job's queue time will fall. For *normal*, the accuracy score was a high 0.95 with a last bin rescheduling accuracy of 0.92. Using 60 minutes bins, the last bin contains jobs with a predicted queue time greater than 4 hours, which makes them the best candidates for dynamic provisioning.

In conclusion, the HGB models developed in the study obtained a sufficiently high R^2 score (0.9, regression) and accuracy (0.92, classification) to be suitable for use in a smart scheduling application.

7. Related Work

Several works have used machine learning approaches to predict queue waiting time and job runtime using HPC job historical data ([11]–[14], [16], [17]). In [12], the authors used machine learning approaches to predict job start time quickly and accurately to help place urgent workloads across HPC machines. They proposed a stochastic method to generate random queue states that capture the machine usage patterns and use that as input for the model. They used a combination of boosted trees classification and regression models from the XGBoost library with the proposed stochastic method, which improved the accuracy significantly. They can accurately predict the job's start time 85% of the time within 60 minutes, 90% of the time within 2 hrs, and 95% of the time within 6 hrs on all three HPC machines' standard queues considered in the paper. Their results are comparable with ours, where the last bin rescheduling accuracy, i.e., jobs with 4 hrs or more queue time, is 91%. In [11], the authors surveyed the runtime prediction studies and recommended a systematic approach to developing a machine learning runtime prediction model. They evaluated their approach on the NREL Eagle HPC dataset. They found that the XGBoost model performs well with a training window of 100 days and a testing window of one day, indicating that the job runtime prediction model needs to be retrained daily. In our study, we found Histogram-based Gradient Boosting regressor and classifier models training on 22 days, testing on 5 days, and validating on future 3 days data give sufficient accuracy for the smart-scheduling use case. The same authors in [14] did another survey of queue time prediction literature complementing their work on the runtime prediction. They investigated how queue time prediction can be improved using job run-time prediction information.

In [13], the authors applied both unsupervised and supervised machine learning techniques for queue time predictions on job historical data collected from real supercomputer. They observed that supervised models performance was dependent on the way the dataset was split. They also proposed an uncertainty quantification approach to compute the reliability of the predictions and find error's distribution.

In [16], the authors proposed to predict job waiting time using Hidden Markov Model. They removed the outliers and then estimated the hidden state parameters via Baum-Welch algorithm. The prediction accuracy was up to 60%

In a recent unpublished work [17], the authors applied supervised learning algorithms - eXtreme Gradient Boosting (XGBoost), Random Forest (RF) and Multi-Layer Perceptron (MLP) on the HPC jobs history data from Theta Cray XC40 and Polaris machines at Argonne National Laboratory to predict job wait times. It uses outlier detection and Principal Component Analysis for efficient data pre-processing and improving the prediction accuracy. It explores relationships between the job characteristics and the job waiting times. For regression analysis, they used the metrics RMSE, MAPE, and R^2 while for classification algorithms, they computed the classification accuracy, learning time and prediction time. They found that tree-based models, specifically XGBoost, outperforms MLP and even RF. The key difference with our approach is that we provide a python framework that does a model grid search with different hyperparameters for several machine learning algorithms. Also, we analyzed the different window sizes for training/testing/future split. We provide a holistic view how such smart scheduling module can be incorporated in a platform like Tapis and can enhance the user experience and their capability to do research more efficiently.

In another work [18] that did not use ML model, the authors proposed a methodology called Binomial Method Batch Predictor (BMBP) that predicts job waiting times in a queue within certain confidence interval. It tries to detect various change points in the history data to make each prediction for those relevant history.

Given the extensive literature concerning queue time prediction for HPC history data, a common pattern emerges that XGBoost or Historgram-based Gradient Boosting model perform better than other models and even outperform deep learning models. In [19], the authors did extensive benchmarks and comparison of tree-based models such as XGBoost and Random Forest, and deep neural networks across 45 generic tabular datasets in multiple settings. They found that tree-based models outperforms deep learning models on medium-sized tabular data (10K). In different experiments, they observed three reasons that could explain the performance gaps between tree-based models and deep learning models - 1) tree-based models are better at learning irregular patterns of the target functions, 2) deep learning models are rotation invariant while tabular data are rotation non-invariant, and 3) deep learning models are sensitive to

Table 1. EVALUATION RESULTS

Queue	# Jobs (Feb-Jul'22)	#Days/W	#Jobs/W	Regression				Classification		
				Alg.	r2Score	MAE	RMSE	Alg.	Acc.	LastbinAcc
skx-normal	169114	30	33822	HGB	0.9	57.85	147.72	HGB	0.92	0.91
				kNN	0.88	44.46	152.21	kNN	0.95	0.93
normal	257053	30	51410	HGB	0.83	30.7	87.86	HGB	0.95	0.92
				kNN	0.83	16.66	83.166	kNN	0.95	0.9355

uninformative features used for training and testing purposes. Our work supports the observation that tree-based models perform better than deep-learning models.

8. Conclusion and Future Work

We presented the challenge of insulating researchers from the intricacies of using cyberinfrastructure by developing smart job scheduling in frameworks like Tapis. We formulated the problem to minimize the time-to-solution by predicting the job queue wait time. We applied machine learning techniques for the prediction and observed that histogram-based gradient boosting methods gave us high accuracy on TACC's Stampede2 data, making it a good candidate to leverage in Tapis's smart scheduling design. In the future, we would like to fine tune our models using different HPC machines' data. Given recent advancements in using large language models (LLMs) on tabular data, we would also like to explore their applicability to our data.

Acknowledgment

Supported by NSF awards #1931439 and #2112606. Special thanks to Constantinos Skevofilax for data curation.

References

[1] J. Stubbs, R. Cardone, M. Packard, A. Jamthe, S. Padhy, S. Terry, J. Looney, J. Meiring, S. Black, M. Dahan, S. Cleveland, and G. Jacobs, "Tapis: An API Platform for Reproducible, Distributed Computational Research," in *Advances in Information and Communication FICC 2021*. Springer International Publishing, 2021, pp. 878–900.

[2] DMTF. (2024, Apr.) All Published Versions of DSP0266. [Online]. Available: https://www.dmtf.org/dsp/DSP0266

[3] Internet Engineering Task Force (IETF). (2024, Apr.) An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3411

[4] J. Stubbs, S. Marru, D. Mejia, J.-P. Navarro, E. Franz, S. Black, D. Wannipurage, S. Pamidighantam, C. Stirm, M. Dahan, M. Pierce, and M. Zentner, "Common resource descriptions for interoperable gateway cyberinfrastructure," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '21. ACM, 2021.

[5] Science Gateways Community Institute (SGCI). (2024, Apr.) SGCI Resource Inventory. [Online]. Available: https://sgci-resource-inventory.readthedocs.io/en/latest/

[6] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra, "Comprehensive resource use monitoring for hpc systems with tacc stats," in *2014 First International Workshop on HPC User Support Tools*, 2014, pp. 13–21.

[7] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[8] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[9] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[10] J. Stubbs, S. Padhy, and R. Cardone. (2024) ml-smart-scheduling. [Online]. Available: https://github.com/tapis-project/ml-smart-scheduling/

[11] K. Menear, A. Nag, J. Perr-Sauer, M. Lunacek, K. Potter, and D. Duplyakin, "Mastering hpc runtime prediction: From observing patterns to a methodological approach," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '23. ACM, 2023, p. 75–85.

[12] N. Brown, G. Gibb, E. Belikov, and R. Nash, "Predicting batch queue job wait times for informed scheduling of urgent hpc workloads," in *Proceedings of the Cray User Group*, Jun. 2022, cray User Group, CUG ; Conference date: 01-05-2022 Through 05-05-2022. [Online]. Available: https://cug.org/cug-2022/

[13] C. Vercellino, A. Scionti, G. Varavallo, P. Viviani, G. Vitali, and O. Terzo, "A machine learning approach for an hpc use case: the jobs queuing time prediction," *Future Generation Computer Systems*, vol. 143, pp. 215–230, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X23000274

[14] K. Menear, K. Konate, K. Potter, and D. Duplyakin, "Tandem predictions for hpc jobs," in *Practice and Experience in Advanced Research Computing 2024: Human Powered Computing*, ser. PEARC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3626203.3670547

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] J.-W. Park, M.-W. Kwon, and T. Hong, "Queue congestion prediction for large-scale high performance computing systems using a hidden markov model," *J. Supercomput.*, vol. 78, no. 10, p. 12202–12223, Jul. 2022. [Online]. Available: https://doi.org/10.1007/s11227-022-04356-z

[17] N. Okafor, B. Lusch, and V. Vishwanath, "Queue wait time prediction in high performancecomputing (hpc) systems (under submission)," 2024.

[18] J. Brevik, D. Nurmi, and R. Wolski, "Predicting bounds on queuing delay for batch-scheduled parallel machines," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 110–118. [Online]. Available: https://doi.org/10.1145/1122971.1122989

[19] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22.