Optimized Hardware Procurement for High Performance Computing Systems

Scott Hutchison
Department of Computer Science
Kansas State University
Manhattan, KS 66505, USA
email: scotthutch@ksu.edu

Daniel Andresen
Department of Computer Science
Kansas State University
Manhattan, KS 66505, USA
email: dan@ksu.edu

William Hsu
Department of Computer Science
Kansas State University
Manhattan, KS 66505, USA
email: bhsu@ksu.edu

Mitchell Neilsen
Department of Computer Science
Kansas State University
Manhattan, KS 66505, USA
email: neilsen@ksu.edu

Benjamin Parsons
High Performance Computing Modernization Program
Engineering Research and Development Center
Vicksburg, MS 39180, USA
email: ben.s.parsons@erdc.dren.mil

***Abstract*—When faced with upgrading or replacing High Performance Computing or High Throughput Computing systems, system administrators can be overwhelmed by hardware options. Servers come with various configurations of memory, processors, and hardware accelerators, like graphics cards. Differing server capabilities greatly affect their performance and their resulting cost. For a fixed budget, it is often difficult to determine what server package composition will maximize the performance of these systems once they are purchased and installed. This research uses simulation to evaluate the performance of different server packages on a set of jobs, and then trains a machine learning model to predict the performance of un-simulated server package compositions. In addition to being orders of magnitude faster than conducting simulations, this model is used to power a recommender system that provides a precision@50 of 92%. This model is further evaluated using 24 days throughout the calendar year, and it achieves a precision@50 of 88%.**

***Index Terms*—HPC; Procurement Optimization; Recommender system; XGBoost.**

I. Preface

This research was originally presented at The Seventeenth International Conference on Advanced Engineering Computing and Applications in Sciences [1]. The results presented there have been expanded upon and further clarified for this publication.

II. Introduction

When faced with upgrading or expanding a High Performance Computing (HPC) or High Throughput Computing (HTC) system, administrators of these systems can be overwhelmed by options. It is a challenging task to get the best performance for a fixed budget. Server capabilities (i.e., number and types of processors, amount of memory, and number and types of Graphics Processing Units (GPU) or other hardware accelerators) greatly affect their costs, and for a fixed spending ceiling, it is desirable to get the "best bang for your buck." For an HPC system, an optimal server package composition is dictated by its typical use. For instance, if many users rely upon a GPU-accelerated application or library, a higher GPU count may be desirable, even if this means fewer servers can be purchased. With many factors to consider, HPC administrators often rely upon their preferences, intuition, and experience to inform procurement decisions. This research uses historical job data from an HPC system, a discrete event simulator (DES), and a machine learning model to power a recommender system, which can help inform a hardware procurement decision. These techniques provide additional information to HPC system administrators about which set of budget-constrained hardware minimizes wait time for users' jobs, and provides quantifiable support for procurement decisions when upgrading or expanding existing HPC infrastructure. The contributions of this work can be summarized as follows:

1) A data set consisting of roughly 12,700 HPC scheduling simulations, each with a different HPC server set
2) An optimized XGBoost regression model for predicting average wait time when given a composition of servers
3) A recommender system with precision@50=92%, which can inform hardware procurement decisions

This paper is laid out as follows: Section III provides additional background on the problem and describes similar work done by others, Section IV provides the methodology and some implementation details, Section V provides details of formulas for metric calculations, Section VI provides the results of the experiments, and Section VII provides additional details on how the recommendations of the system were evaluated across a wider time frame. Section VIII evaluates the performance of a model trained using all available data, and Section IX provides our final conclusions.

III. Background and Related Works

The Open Science Grid (OSG) [2] [3] is a worldwide collaboration that offers distributed computing for scientific research. In the central United States, one of the organizations contributing resources to the OSG is the Great Plains Augmented Regional Gateway to the Open Science Grid (GP-ARGO) [4]. In part, GP-ARGO receives funding through governmental grants. These grants are often used to procure new equipment to expand or improve the capabilities of GP-ARGO's participating organizations. Consequentially, there is a fixed budget ceiling for HPC equipment procurement, and the administrator's goal is to purchase new equipment that will maximize computational performance for our typical applications while ensuring costs remain under the fixed grant budget. The research question for this work is as follows: for a planned HPC expansion, can experimental simulation provide an optimal set of hardware under a given budget that will minimize job wait time?

The challenge of optimal hardware procurement is not exclusive to our organization. Similar work was done by Evans et al. [5]. They collected benchmarks for various software applications on different hardware to optimize the ratio of Central Processing Unit (CPU) and GPU architectures for HPC jobs. Their work is similar to ours, but we took a different approach by using a scheduling simulator to evaluate the performance of a set of jobs that were actually submitted to an HPC system. We are solving a very similar problem as Evans et al., but using a different approach to arrive at an optimal hardware configuration.

Other researchers have attempted to optimize for a particular application, such as the work Kutzner et al. [6] did to improve the utilization of GPU nodes when using GROMACs. Although these techniques are not without their merits for HPC systems that run a large number of homogeneous applications, users of the GP-ARGO HPC systems run a wide variety of jobs and applications. A more broad scheduler-based optimization was more appropriate for our application.

Various public HPC workloads exist [7], and have been used by HPC researchers in the past. However, as we are attempting to identify and evaluate new hardware for a specific HPC system, log data from that HPC system was utilized as the workload for this research.

Different scheduling applications like SLURM, HTCondor, or PBS, operate on HPC systems and perform the function of assigning HPC resources to jobs. This job-to-machine-assignment task is as an extension of the online bin packing problem [8]. For the bin packing problem, the goal is to pack a sequence of items with sizes between 0 and 1 into as few bins of size 1 as possible. Each job specifies the resources requested (the object sizes), and each HPC machine has a certain amount of available resources (the bins with their respective sizes). The scheduler is given the task to meet job requirements by assigning them to HPC nodes (pack the objects into the available bins) as efficiently as possible. This is an online problem as new jobs are submitted over time to the scheduler. The best fit bin packing (BFBP) algorithm has been shown by Dosa and Sgall [9] to use at most $\lfloor 1.7OPT \rfloor$ bins, ensuring this algorithm will provide a reasonably close to optimal average wait time when it is used as an HPC job scheduling algorithm.

Algorithm 1 Best Fit Bin Packing Scheduling

```
1: while The simulation is incomplete do
2:     if Some job in the queue can be executed on some machine then
3:         Find the (job, machine) pairing that results in the fewest remaining resources for some machine. Begin executing that job on that machine.
4:     else
5:         Advance simulation time until a new job is submitted or a running job ends, whichever is sooner.
6:         Queue submitted jobs and stop ending jobs.
7:     end if
8: end while
```

Fig. 1. Pseudocode for the best fit bin packing algorithm

Since scheduling algorithms vary between applications, most being highly customizable, and others being proprietary, a discrete event simulator utilizing the BFBP algorithm served as a stand-in for our scheduling application in an attempt to make it more universally applicable. The BFBP scheduling algorithm is described in Figure 1.

Although various HPC simulators have been used for similar research, such as SimGrid [10], GridSim [11], or Alea [12], this experiment needed a simple discrete event simulator using the BFBP scheduler. The simulators mentioned above were either deemed overly complex for our purposes, or they failed to allow for the three limiting resources (memory, CPUs, and GPUs) we were interested in investigating. An HPC scheduler simulator was also considered, such as the Slurm simulator developed at SUNY University in Buffalo [13]. Although this option was investigated further, scaling a job's actual duration from the log data to the new machine once it is assigned to a machine was challenging. As such, a custom discrete event simulator was developed and utilized for this research. The simulator allows for three resource constraints in each machine: memory, CPUs, and GPUs. It is fairly lightweight, fast, and easy to understand.

A significant consideration when evaluating new server hardware is the performance increase newer technology or architectures can provide. Using log data, we know how long a job took on a machine with known hardware. Since the specifications for the new hardware under consideration are also known, the actual duration of the jobs from the historic log data was scaled using base performance of the processor as reported by SPEC CPU2017 benchmark, second quarter, 2023 [14].

Knowing how a particular job performed on one set of hardware and estimating how it will perform on some other hypothetical set of hardware is challenging. Sharkawi et al. [15] successfully used a similar SPEC benchmark to estimate the performance projections of HPC applications. Other researchers, like Wang et al. [16] have pointed out that these benchmarks fail to account for all the variables affecting job resource utilization and should be avoided. Although CPU performance is not the only factor by that we could have scaled job duration, and perhaps it is not the best factor by which to scale, it worked well for our purposes. The discrete event simulator was implemented such that the scaling factor could be easily changed if other researchers should find a different factor more relevant to their situation.

Various metrics are typically used when evaluating the performance of HPC scheduling algorithms. Some of these are average wait time, HPC utilization, average turnaround time, makespan, throughput, etc. Which metric is used depends on the application and function of the HPC system, and different organizations may value one metric over another. The metric used for this research was average wait time, or the average number of seconds each job spent waiting in the job queue for execution on HPC resources. We presume that the same techniques could be applied by other researchers using a different metric, should they prefer a different one.

This research relied upon a regression model where: given the total CPUs, total memory, and the total GPUs for a composition of servers, the regression model will predict the average jobs wait time for the representative set of jobs. Various regression techniques were tried, but Extreme Gradient Boosting (XGBoost) [17] was the most effective of those tried. XGBoost is a scalable, distributed, gradient-boosted, decision tree machine learning library. It relies upon supervised machine learning, decision trees, ensemble learning, and gradient boosting. Similar to a random forest, multiple decision trees are created for the regression task, and these trees each make predictions of the average wait time given the three inputs (total server package CPUs, memory, and GPUs). The results from the multiple trees are combined via a weighted sum, and they are "boosted" by generatively adding new decision trees. The error of the objective function is minimized by gradient descent during the training process, resulting in quick convergence and accurate prediction results.

Recommender systems power a variety of applications like search engines and music recommendation systems. First, the "hits" for the system must be defined. Hits are the elements from the data set that are relevant to the user's search. Next, the user specifies the number of recommendations, k, that they would like to receive. If the recommender system is precise, a large portion of the k items returned will be hits.

IV. Methodology

The general plan for optimizing a hardware package for our fixed budget can be summarized as follows:

1) Receive vendor quotes with potential server options.
2) Generate potential server combinations to purchase under the specified budget which meet our procurement requirements.
3) Identify a typical set of jobs representing the workloads typically submitted to our HPC system.
4) Conduct simulations using a subset of the server packages to schedule the representative job set and compute metrics to determine their performances.
5) Use machine learning to train and refine a regression model that can predict the performance of un-simulated server combinations.
6) Develop a recommender system using the machine learning model and quantitatively evaluate its performance
7) Subjectively evaluate the recommended server packages and make a more informed procurement decision.

This pipeline is illustrated by Figure 2. First, we generate all possible combinations of servers we can purchase under our budget. Next, we uniformly sample 10% of these by selecting every tenth server combination and we use the DES to simulate the execution of a chosen set of jobs. We then use XGBoost to develop and train a regression model that will map a sever package's total CPUs, memory, and GPUs to the predicted average wait time that these jobs will experience. We use the regression model to predict the average wait time of the sampled server combinations, sort them by the predicted wait time, and return the top k recommended server sets to the user. These recommendations can be quantitatively evaluated, as the actual average wait time has been simulated. Next, we can use the same regression model and recommender system to make predictions on the entire set of servers, and we can summarize them and subjectively evaluate them, as 90% of the have not be simulated and their actual average job wait time is unknown.

Finally, the recommendations of the system were simulated using workloads from 24 days across a calendar year to determine it the recommended server sets continued to be effective when faced with the varied workloads the HPC system experienced throughout the year.

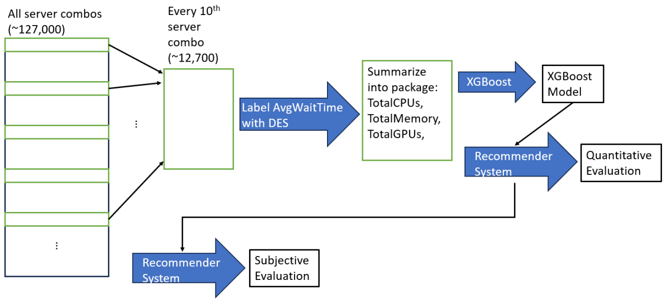
All server combos (~127,000)
Every 10th server combo (~12,700)
Label AvgWaitTime with DES
Summarize into package: TotalCPUs, TotalMemory, TotalGPUs,
XGBoost
XGBoost Model
Recommender System
Quantitative Evaluation
Recommender System
Subjective Evaluation

Fig. 2. A pictorial representation of the methodology for this research.

A. *Generate Server Options*

To begin, we received several vendor quotes specifying the costs and capabilities of 21 potential servers to purchase. When considering upgrade options, we typically separate servers into one of three categories: compute nodes, big memory nodes, or GPU nodes. A compute node typically has a large number of processor cores, a moderate amount of memory, and no GPU. A big memory node will have a large amount of

TABLE I. SERVER CAPABILITIES AND COSTS UNDER INVESTIGATION

Node type	*Distinct nodes considered*	*Memory range per node*	*CPUs range per node*	*GPUs per node*	*Cost range per node*
Compute	4	256-512 Gb	24-64 cores	0 GPUs	\$6k-\$10k
Big memory	2	1024 Gb	24-64 cores	0 GPUs	\$11k-\$13k
GPU	15	256-1024 Gb	24-64 cores	1-8 GPUs	\$14k-\$100k

memory with a moderate amount of CPU cores and no GPU. A GPU node is any node that has a GPU. Table I lays out the options we received from several different vendors. The procurement budget was fixed at $1 million, and all possible server combinations were generated in the following way:

- Separate servers into three categories: compute nodes, big memory nodes, and GPU nodes.
- Choose all combinations of one node from each category.
- Determine all quantities of the three node types under a given budget such that there is at least one GPU node and there is not enough funding remaining to purchase another node.

In our selected job set, many jobs requested GPUs as a resource. These jobs would automatically fail if at least one GPU node were not included in a potential server package. Roughly 127,000 different server combinations met these requirements. Table II provides an illustrative example of how the server combinations were generated. Many server options and packages were omitted from the table for the sake of brevity.

B. Identify a Representative Set of Jobs

One typical days' worth of submitted jobs (roughly 16,000 jobs) was subjectively pulled from the log data of the local HPC system. As with most HPC systems, jobs were submitted in a bursty manner, and variety of resources were requested. Figure 3 and Table III display some descriptive statistics and information about the jobs used for this portion of this research.

C. Job Duration Scaling

The submitted jobs were scaled using the base performance of the processor on the SPEC CPU2017 benchmark suite. The requested duration was not modified, but the actual duration of each job was calculated using the following formula:

$$\text{New duration} = \frac{\text{logged duration} * \text{logged processor performance}}{\text{new processor performance}}$$

D. Discrete Event Simulator

Since there are many different applications for scheduling jobs on HPC systems, the discrete event simulator using the BFBP scheduling algorithm acted as a generic substitute for the scheduling application for our HPC system. What was needed was a method for determining the average job wait

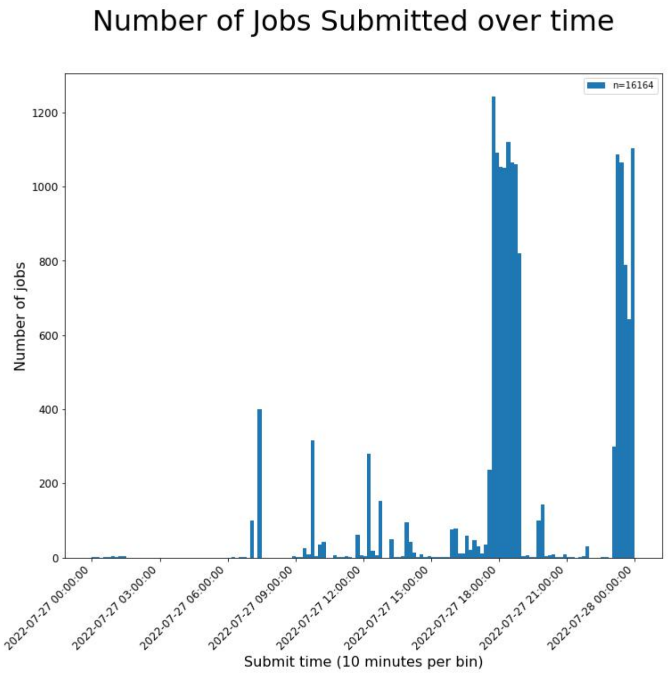

Fig. 3. The number of jobs submitted over time for the selected day

time for selected jobs on specified hardware. Other simulators could have been used, but for this research, a discrete event simulator was implemented in Python that provides the following functionality:

- A global clock to keep track of simulation time.
- Several queues, priority queues, or lists to track jobs as they progress through the execution process: future jobs, queued jobs, running jobs, completed jobs, and unrunnable jobs.
- Jobs and machines are specified using comma separated value (csv) files, which is loaded prior to the simulation.
- Machines have three limiting resources: available memory, CPUs, and GPUs.
- Jobs are specified with the following attributes: submit time, actual duration, and requested duration, memory, CPUs, and GPUs. Jobs track their start time and end time as the simulation progresses to allow for metric calculation.
- Job end time is set when the job starts running as the job start time plus the job actual duration.
- When a job starts running on a machine, that machine's available resources are decremented by the resources requested by the job. Conversely, when a job completes, the machine executing it has its available resources increased by the amount requested by the ending job.
- Jobs with a submit time greater than the current global clock reside in the future jobs priority queue.
- Jobs with a submit time less than or equal to the current global clock, but not yet assigned to a machine, reside in the job queue.
- Jobs that have begun their execution and have an ending

TABLE II. GENERATED SERVER COMBINATIONS

ComputeNode1, $6,960 ea.	*BigMemNode1, $11,112 ea.*	*GPUNode1, $14,730 ea.*	...	*Package Cost*	*Funds Remaining*
141	0	1	...	$996,090	$3,910
139	1	1	...	$993,282	$6,718
138	2	1	...	$997,434	$2,566
⋮	⋮	⋮	⋱	⋮	⋮
0	1	67	...	$998,022	$1,978

TABLE III. DESCRIPTIVE STATISTICS FOR THE POOL OF SELECTED JOBS

	Requested Mem (in Gb)	*Requested CPUs*	*Requested GPUs*	*Requested Duration (in hours)*	*Actual Duration (in hours)*
Mean	5.12	4.75	0.002	2.82	2.27
Std Dev.	16.73	3.33	0.055	1.02	13.67
Min	1	1	0	0	0
Max	800	64	4	11.20	11.20

time less than the current global clock, reside in the running jobs priority queue.

- Jobs with an ending time less than or equal to the current global clock reside in the completed jobs list.
- If no node in the cluster has adequate resources to run a particular job, that job is moved to the unrunnable jobs list.
- In the event that no queued jobs can run on available resources, the simulation time "fast forwards" to the next event: either job submission or job ending.
- Jobs in the job queue are run as soon as there are available resources and are chosen using the best fit bin packing scheduling algorithm described in Algorithm 1.
- Actual job duration from logged job data can be scaled to allow for hardware improvement with newer hardware.

E. Machine Learning

Although each simulation completed fairly quickly, requiring no more than 30 minutes each, this particular combination of server quotes yielded roughly 127,000 combinations that need to be evaluated. To reduce the computational requirement, every tenth line from the file with the server combinations was sampled, and roughly 12,700 simulations for these server packages were completed in parallel using HPC resources. By sampling from the generated server packages uniformly, various quantities of each server under consideration were included in the simulated data. Each server package was summarized into the package total memory, total CPUs, and total GPUs, by summing the resources of every machine comprising the package. The average wait time for the simulation served as the label for each package. Using five fold cross validation, an XGBoost regression model was trained using training data. The regression model was evaluated using root mean squared error (RMSE) on the test data. An accurate regression model enabled the prediction of the average wait time for unsimulated server combinations and saved countless hours of additional simulation.

F. Recommender System

In our case, a hit was defined as a server combination with an average wait time in the lowest 5% of simulated combinations (or 632 hits out of the ∼12,700 simulated server combinations). The value of k was varied to evaluate the performance of the recommender system. Then, once confidence was gained that our recommender system was functioning properly, it was used to recommend systems from the entire server combination pool of 127,000 server combinations. The recommendations were summarized and evaluated subjectively before arriving at a final procurement decision.

G. Simplifying Assumptions

The current nodes comprising the HPC system were not added to the set of nodes simulating the selected jobs. The benefit current nodes would provide to the new servers under investigation would be common to all.

Any additional equipment required to install and operate the new servers (e.g., networking hardware, additional cooling equipment, server racks, power infrastructure, etc.) were not deducted from the total procurement budget. It was thought that these costs would be a relatively fixed regardless of the server package chosen. The same analysis described in this research could be done by reducing the total budget by the cost of additional hardware and then completing the analysis with a reduced budget.

V. EVALUATION

Pearson's Correlation Coefficient [18] determined the extent of the correlation between the total memory, CPUs, and GPUs of a package and the average wait time. This coefficient provides a value between -1 and 1, where values closer to -1 or 1 indicate that the feature and the label are more strongly correlated. A coefficient of 0 indicates no correlation.

Wait time was calculated by analyzing the completed jobs output from each simulation. The wait time for each job was the number of seconds from the time the job was submitted until it began. For N jobs, the average wait time was calculated as follows:

$$\text{AvgWaitTime} = \frac{\Sigma_{i=0}^{N}(\text{Start Time}_i - \text{Submit Time}_i)}{N}$$

Root Mean Squared Error was utilized for regression model evaluation calculated according to the following formula:

$$\text{RMSE} = \sqrt{\frac{\Sigma_{i=0}^{N}(\text{actual wait time}_i - \text{predicted wait time}_i)^2}{N}}$$

The performance of the final recommender system was evaluated using precision@k, recall@k, and F1@k. In general, precision@k is the proportion of recommended items in the top-k set that are relevant, and recall@k is the proportion of relevant items found in the top-k recommendations. F1@k is the harmonic mean of precision@k and recall@k, which simplifies them into a single metric. They were calculated according to the following formulas:

$$\text{Precision@k} = \frac{(\text{\# of recommended items @k that are relevant})}{(\text{\# of recommended items @k})}$$

$$\text{Recall@k} = \frac{(\text{\# of recommended items @k that are relevant})}{(\text{total \# of relevant items})}$$

$$\text{F1@k} = \frac{(2 * \text{precision@k} * \text{recall@k})}{(\text{precision@k} + \text{recall@k})}$$

VI. Results

The correlation of features, the performance of the regression model and the recommender system, and some analysis about the recommended server compositions are described below.

A. Feature Correlation

The correlation between the features and the labels is shown in Table IV. For this set of jobs, the total CPUs in a server package were most strongly correlated to the average wait time. For the chosen jobs, the more CPUs a package had, the lower its average wait time.

Since we are constrained by our available budget of $1 million, choosing to buy one type of node over another is a zero-sum game. The more GPU nodes we purchase, and the more GPUs there are per node, the fewer compute nodes or big memory nodes we are able to afford. This is indicated by the positive correlation between GPUs and the average wait time.

TABLE IV. Pearson Correlation Coefficients

	TotalMem	*TotalCPUs*	*TotalGPUs*	*AvgWaitTime*
TotalMem	1.00	0.14	−0.54	−0.23
TotalCPUs	0.14	1.00	−0.42	−0.70
TotalGPUs	−0.545	−0.42	1.00	0.44
AvgWaitTime	−0.23	−0.70	0.44	1.00

B. Regression Model

The XGBoost regression model had a RMSE = 150.13 seconds, indicating that the total memory, CPUs, and GPU features made excellent predictors for the average wait time for these jobs when simulated with the discrete event simulator. The predicted vs. simulated wait time is shown in Figure 4. If the regression model were perfect, all these points would lie upon the $y = x$ line, and it is clear that this model does a good job at predicting the average wait time for a given composition of servers.

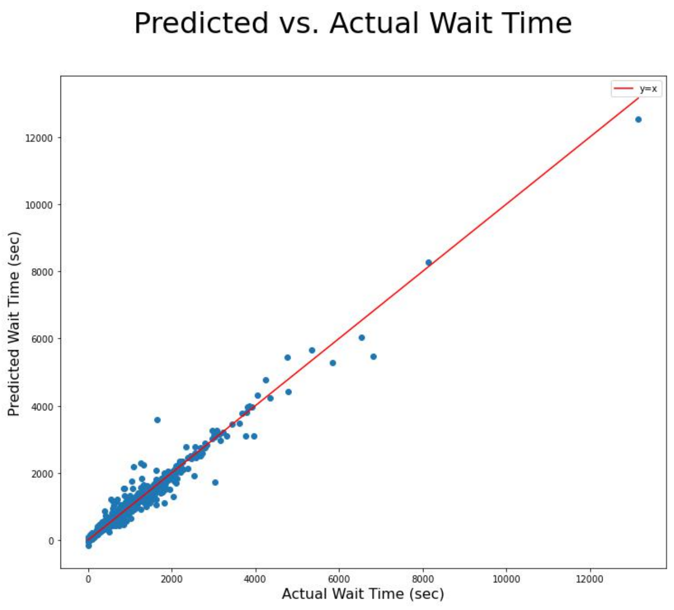

Fig. 4. The predicted vs. simulated wait times showing the accuracy of our regression model.

C. Recommender System

The regression model was used to predict the 12,700 labeled simulations, and their precision@k, recall@k, and F1@k for various values of k are displayed in Table V. The goal was to reduce the number of possibilities from roughly 127,000 different possible combinations of servers down to a reasonable number that could be evaluated by an HPC system administrator and have a large percentage of the recommended server combinations be hits (among the best 5% of server combinations with the lowest average wait times). Although precision@10 was 100%, it is thought that seeing more server package options would allow system administrators a wider variety from which to choose. A system administrator could easily and quickly review up to 50 recommendations ($k = 50$), and more than 46 out of 50 of these recommendations returned by this system (92%) would be top performing server combinations, which is excellent. Recall@k when k is less than the number of total hits (632 hits total) is unfairly penalized, but the recall@k above 632 is also excellent. When $k = 1,000$, the recall@1000 = 91%, meaning the recommender system successfully retrieved 91% of the top 5% performing server packages when returning less than 1% of the 127,000 different options.

TABLE V. Precision@k and Recall@k for Test Data

k value	*Precision@k*	*Recall@k*	*F1@k*
10	1.00	0.02	0.03
50	0.92	0.07	0.13
100	0.81	0.13	0.22
500	0.74	0.59	0.66
632	0.72	0.72	0.72
1000	0.58	0.91	0.71

D. Recommended Compositions

Beyond looking at the individual server compositions recommended, we wanted to draw some conclusion about the types and quantity of nodes that the recommender system returned. The sum of the server quantities for the top 50 recommendations can be found in Table VI. Compute nodes with the larger number of cores were vastly preferred, and the recommender system did not recommend spending additional funds on more memory for the compute nodes. Additionally, the recommender system preferred the cheaper big memory node with fewer cores. Finally, for our typical workload, the recommender system did not recommended purchasing a large number of GPUs per GPU node, instead recommending servers with 2 GPUs per server most often. As shown in Figure 5, the recommender system suggests spending on average 58% of our total budget on compute nodes, 8% on big memory nodes, and 34% on GPU nodes.

TABLE VI. RECOMMENDATIONS DRAWN FROM MODEL PREDICTED RESULTS

Node Type	*Node Description*	*Sum of Servers Across Top 50*
Compute Nodes	Low Cost CPU w/ 256Gb	232
	Low Cost CPU w/ 512Gb	0
	High Cost CPU w/ 256Gb	3,467
	High Cost CPU w/ 512Gb	0
Big Memory Nodes	Low Cost CPU w/ 1024Gb	232
	High Cost CPU w/ 1024Gb	111
GPU Nodes	2 GPUs in one server	732
	4 GPUs in one server	267
	8 GPUs in one server	0

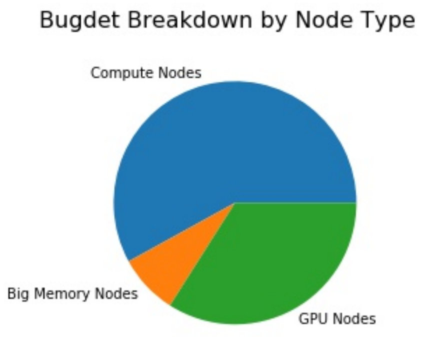

Fig. 5. The recommended budget breakdown by node type.

A boxplot showing the simulated average job wait time of the top 5% of recommended server sets (or k=638) is shown in Figure 6. It is clear that the recommender system was able to retrieve and recommend server sets that performed well on the representative set of jobs.

VII. EVALUATION OF THE GENERALIZATION OF THE APPROACH

The previously described regression model was developed using the results when using a single, representative workload

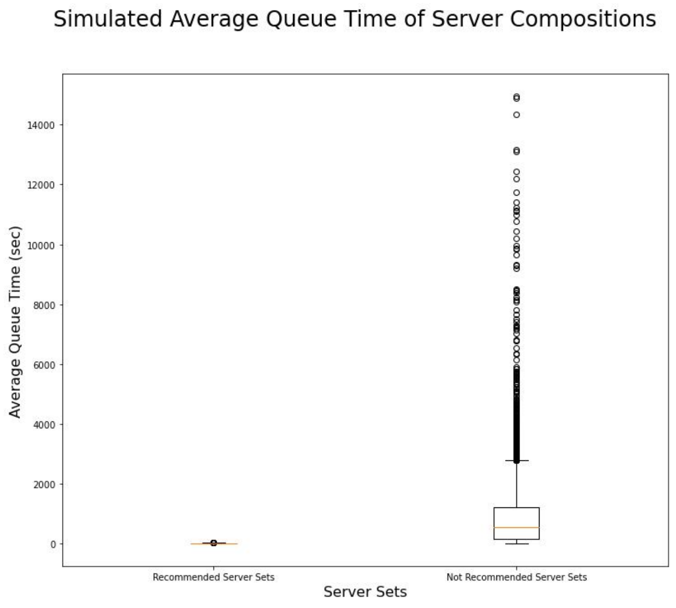

Fig. 6. The average job wait time for the top 5% recommended server sets vs. the bottom 95%.

of one days' worth of jobs from the local HPC system. For this technique to be viable, it must be demonstrated that the recommended server packages would perform well not only for the representative day, but also for many different days of HPC activity. To investigate this further, the following methodology was used:

- Subjectively pull log data an additional 24 days across the year (2 days per month)
- Use the discrete event simulator to simulate the execution of the workloads for the same subset of 12,700 server packages
- Identify the top 10% of server compositions with the lowest average wait time for each day
- Evaluate the performance of the initial recommended server sets using the additional labeled data

This computation was done in parallel using HPC resources and involved over 150,000 CPU hours.

A. Generalized Results

To begin, 24 different days of HPC log data from throughout the year were chosen subjectively (2 days per month). These days were scheduled using the DES using the roughly 12,700 server combinations that were originally used to train the regression model. See Figure 7 for the average wait times for each of the days simulated. Since the job characteristics for each day were different, this caused a re-ordering of the "hits" for each day. For instance, if a day had many GPU jobs, server combinations with more GPUs would have lower average job wait times. Each day's hits were defined as the server set whose average job wait time was in the lowest 10% for that day. By counting the number of days across the year for which that server combination was in the top

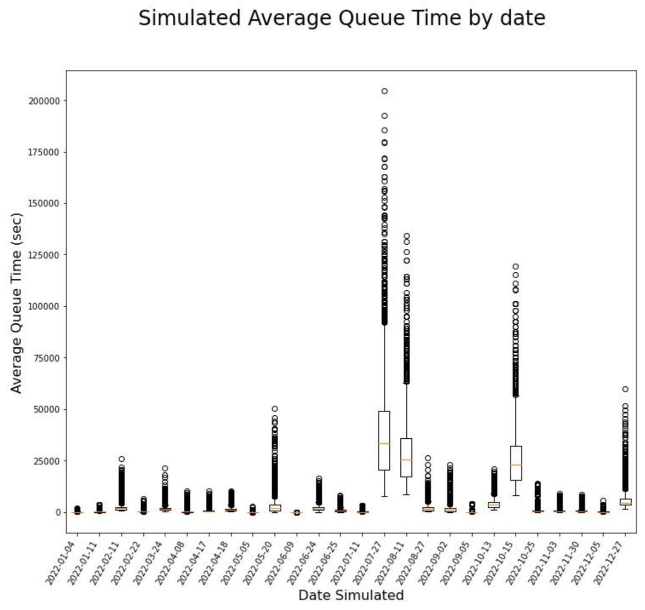

Fig. 7. Boxplots of the average wait time for each of the days simulated throughout the year.

10% of performant server combinations, we were able to determine which server combinations were an overall "hit" for the recommender system. A hit threshold of *num_hits*> 6 was found to produce good results, meaning that for 7 or more days of the 25 labeled days throughout the year, the server combination was in the top 10% of performant server packages. The results can be found in Table VII.

TABLE VII. PRECISION@K AND RECALL@K WHEN HIT THRESHOLD >6

Hit Threshold	k	precision@k	recall@k	F1@k
>6	10	1.00	0.005	0.01
>6	50	0.88	0.02	0.04
>6	100	0.83	0.04	0.08
>6	500	0.93	0.23	0.36
>6	1000	0.89	0.43	0.58
>6	2046	0.75	0.75	0.75
>6	5000	0.41	1.00	0.58
>6	10000	0.2	1.00	0.34

If $k = 50$, the recommender system achieves a precision@50=88%, which is slightly lower than the precision@50=92% when the day was evaluated on the same day on which it was trained. Again, 50 recommendations is thought the be an easily human parsable amount which can be compared and evaluated by system administrators for purchase. In other words, given the top 50 recommendations returned to the user, 88% of them would be in the top 10% of performant server sets for 7 out of the 15 days throughout the year which were evaluated.

Increasing the hit threshold reduces the number of total hits that the recommender system can find, and consequentially lowers the precision@k. For instance, the threshold mentioned in Table VII required a server set to be among the top 10% for seven or more days out of the 25 days simulated. In this case, there were 2,046 overall "hits" out of the Ĩ2,700 total server sets whose performance was measured. If the hit threshold is raised to 20, meaning 21 or more days found these server combinations in the top 10% of performing server sets, there are only 70 total hits for the recommender system to find. Figure 8 shows how precision@k degrades when the hit threshold is raised. When the hit threshold is raised to 20, the recall@500 is 67%, meaning that the top 500 results returned by recommender system contained 67% of the 70 hits that were found. Table VIII shows the results at this hit threshold. Though these results are less promising, there were relatively few server sets that performed well for this many days throughout they year. Using the recommender system trained on days' worth of representative jobs saved approximately 150,000 hours worth of computation time conducting the simulations on the various jobs submitted throughout the year.

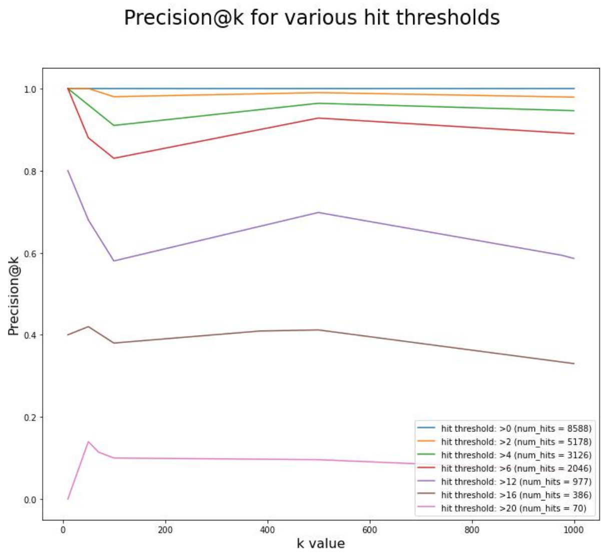

Fig. 8. The precision@k as the hit threshold is varied.

TABLE VIII. PRECISION@K AND RECALL@K WHEN HIT THRESHOLD >20

Hit Threshold	k	precision@k	recall@k	F1@k
>20	10	0.00	0.00	0.00
>20	50	0.14	0.10	0.12
>20	70	0.11	0.11	0.11
>20	100	0.10	0.14	0.12
>20	500	0.10	0.67	0.17
>20	1000	0.07	1.00	0.13
>20	5000	0.01	1.00	0.03
>20	10000	0.01	1.00	0.01

B. Time Savings for this Technique

Each simulation took around 30 minutes to complete, but they were conducted in parallel using HPC resources. The roughly 12,700 server compositions simulated to train the regression model took over 6,000 compute hours to complete. Each of the 24 additional days took another 6,000+ hours, for a total of over 150,000 compute hours required to validate the recommendations across a representative sample throughout the year. An exhaustive search of all 127,000 server combinations for a single representative days' worth of jobs would have taken over 60,000 compute hours, and would have yielded a definitive answer on which server combination would have performed best on a single representative days' worth of jobs. Validating this across 24 days across a calendar year would have required over 1.5 million compute hours on HPC resources. The recommender system built using regression from a subset of the possible servers required a 99.96% decrease in the time required for computation while still achieving a precision@50 of 92%. This model achieved a precision@50 of 88% when using the threshold that for 6 or greater days, the server compositions had the lowest 10% average job wait time for each day. The additional validation step of computing 24 days across the year could even be omitted, as the results from the original trained model did quite well across the year.

VIII. Training with All Data

Though we have shown it is sufficient to train using a single day's worth of representative jobs, we obtained simulated average wait times for jobs for 25 days throughout the year. We wanted to explore the performance of a recommender system trained on all data gathered and compare and contrast its performance with the recommender system described above. For each of the 25 days simulated, the average wait time varied depending on the jobs which were submitted on those days. Figure 7 shows boxplots of the average wait time by day depicting the these variations. As such, these values were scaled using min-max normalization prior to regression using the following formula:

$$Normalized\_value = \frac{(actual\_value - min\_value)}{(max\_value - min\_value)}$$

Performing this normalization across each day transforms the average wait time values into a a unitless value between 0 and 1 where values closer to zero represent the best performing server sets with the lowest average wait time. Though the predictions by the regression model will no longer predict the number of seconds of average wait time a server compositions is expected to to have, predicted lower values still represent server packages with lower expected average wait time for jobs. The regression model was not as accurate as when training using a single representative set of jobs, as depicted in Figure 9. Again, if the regression model were perfect, all the predicted vs. actual values would lie upon the $y = x$ line of the graph. Though this model using normalization does not appear to be as good as the previous one, some loss of precision is expected when normalizing in this manner. We can still use the regression model to power a recommender system and evaluate its performance.

Table IX shows the results of the normalized model when the hit threshold is greater than 16. Using the same $k = 50$ value from before, we achieve a precision@50 of 94%, which is thought to be excellent. In other words, 47 out of the top 50 recommendations returned by this model will be among the top 10% of performant server combinations for 17 or more days throughout the year. Though this model does slightly better than the model trained on one representative days' worth of jobs, it took 24 times more computation to provide the data to train it. Though the original model trained using a single day achieved a lower precision, it was still able to return good results across the year and required substantially less computation time.

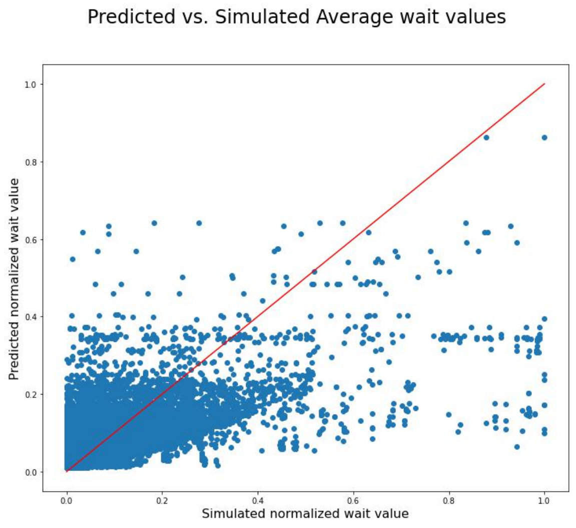

Fig. 9. The predicted vs. simulated normalized average wait time values.

TABLE IX. Precision@k and Recall@k when Hit Threshold >16 for Normalized Model

Hit Threshold	k	precision@k	recall@k	F1@k
>16	10	1.00	0.026	0.05
>16	50	0.94	0.12	0.22
>16	100	0.78	0.20	0.32
>16	386	0.61	0.61	0.61
>16	500	0.54	0.70	0.61
>16	1000	0.32	0.83	0.46
>16	5000	0.08	1.00	0.14
>16	10000	0.04	1.00	0.07

The results of summing the top 50 recommended server sets can be found in Table X. These differ slightly from the recommendations of the model trained using a single representative days' worth of jobs. In both models, the more expensive compute node with more cores was preferred, however the model trained on all the days prefers the compute node with more memory. The model trained on a single day preferred the cheaper big memory node, and the model trained on all the days preferred the more expensive one. Both models preferred GPU nodes with fewer GPUs. Though they differ slightly in the nodes types and quantities they prefer, they are fairly close with their recommendations, and it is thought that the original model using only a single representative day's worth of jobs would provide adequate enough recommendations for a HPC system administrator to evaluate and arrive at a good server combination to purchase.

TABLE X. Recommendations Drawn from Normalized Model Predicted Results

Node Type	*Node Description*	*Sum of Servers Across Top 50*
Compute Nodes	Low Cost CPU w/ 256Gb	0
	Low Cost CPU w/ 512Gb	0
	High Cost CPU w/ 256Gb	1,667
	High Cost CPU w/ 512Gb	1,764
Big Memory Nodes	Low Cost CPU w/ 1024Gb	34
	High Cost CPU w/ 1024Gb	751
GPU Nodes	2 GPUs in one server	341
	4 GPUs in one server	108
	8 GPUs in one server	48

IX. Conclusions

We began with roughly 127,000 different possible server packages that could have been purchased under our budget. We uniformly sampled 10% of these, leaving us with roughly 12,700 different server packages. We chose a representative days worth of jobs from local HPC log data, and simulated the scheduling of these jobs on the 12,700 server packages, so we could calculate the average jobs wait time for a given composition of servers. By developing an XGBoost regression model, we were able to predict the average job wait time for the unsimulated server compositions. Finally, we were able to verify that the recommender system made good recommendations by choosing different sets of jobs spaced throughout the year and simulating the scheduling of different job workloads using those server sets. An administrator considering purchase options has an 88% chance of selecting a top performing server packaged under their budget if they were to choose one from the top 50 recommendations returned by the recommender system. By simulating the performance of a small minority of server packages, our recommender system was able to make excellent recommendations.

The most benefit from using this system comes from the time saved doing simulations. The roughly 127,000 initial server combinations could be effectively summarized by simulating only 10% of them on a single representative set of jobs, and it proved unnecessary to conduct simulations for days spaced throughout the year. This was done for the research in order to evaluate the performance of the recommender system, and explore its feasibility when used to optimize hardware procurement for HPC systems.

This recommender system is not intended to replace the expertise of HPC administrators when it comes to decisions for hardware procurement. It is our hope that this tool can provide a data-driven technique that will help narrow the search space with which administrators are confronted when they make procurement decisions. Returning to the research question: experimental simulation coupled with a regression model enabled a recommender system to return server compositions under a given budget with low average wait times with a precision@50 of 92%. Additionally, the discrete event simulator, job data set, machine learning code, and recommender system code are released under the GPLv3 license should other researchers find it useful (https://github.com/shutchison/Optimal-Hardware-Procurement-for-a-HPC-Expansion).

References

[1] S. Hutchison, D. Andresen, W. Hsu, M. Neilsen, and B. Parsons, “Optimized hardware configuration for high performance computing systems,” in *Proceedings of the 17th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2023)*, International Academy, Research, and Industry Association, 2023.

[2] R. Pordes *et al.*, “The open science grid,” in *J. Phys. Conf. Ser.*, vol. 78 of *78*, p. 012057, 2007.

[3] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein, “The pilot way to grid resources using glideinwms,” in *2009 WRI World Congress on Computer Science and Information Engineering*, vol. 2 of *2*, pp. 428–432, 2009.

[4] “The great plains augmented regional gateway to the open science grid.” https://gp-argo.greatplains.net/. Accessed 2023-01-18.

[5] R. T. Evans, M. Cawood, S. L. Harrell, L. Huang, S. Liu, C.-Y. Lu, A. Ruhela, Y. Wang, and Z. Zhang, “Optimizing gpu-enhanced hpc system and cloud procurements for scientific workloads,” in *International Conference on High Performance Computing*, pp. 313–331, Springer, 2021.

[6] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller, “More bang for your buck: Improved use of gpu nodes for gromacs 2018,” *Journal of Computational Chemistry*, vol. 40, no. 27, pp. 2418–2431, 2019.

[7] D. G. Feitelson, D. Tsafrir, and D. Krakov, “Experience with using the parallel workloads archive,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014.

[8] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[9] G. Dósa and J. Sgall, “Optimal analysis of best fit bin packing,” in *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41*, pp. 429–441, Springer, 2014.

[10] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2899–2917, June 2014.

[11] R. Buyya and M. Murshed, “Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing,” *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[12] D. Klusáček, M. Soysal, and F. Suter, “Alea–complex job scheduling simulator,” in *Parallel Processing and Applied Mathematics: 13th International Conference, PPAM 2019, Bialystok, Poland, September 8–11, 2019, Revised Selected Papers, Part II 13*, pp. 217–229, Springer, 2020.

[13] N. A. Simakov, R. L. DeLeon, M. D. Innus, M. D. Jones, J. P. White, S. M. Gallo, A. K. Patra, and T. R. Furlani, “Slurm simulator: Improving slurm scheduler performance on large hpc systems by utilization of multiple controllers and node sharing,” in *Proceedings of the Practice and Experience on Advanced Research Computing*, pp. 1–8, 2018.

[14] “Second quarter 2023 spec cpu2017 results,” 2023. https://www.spec.org/cpu2017/results/res2023q2, Accessed on June 14, 2023.

[15] S. Sharkawi, D. Desota, R. Panda, R. Indukuru, S. Stevens, V. Taylor, and X. Wu, “Performance projection of hpc applications using spec cfp2006 benchmarks,” in *2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–12, IEEE, 2009.

[16] Y. Wang, V. Lee, G.-Y. Wei, and D. Brooks, “Predicting new workload or cpu performance by analyzing public datasets,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, pp. 1–21, 2019.

[17] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[18] K. Pearson, “Note on regression and inheritance in the case of two parents,” *Proceedings of the Royal Society of London*, vol. 58, no. 347-352, pp. 240–242, 1895.