Dynamic optimization of provider-based scheduling for HPC workloads

Dynamic Optimization of Provider-Based Scheduling for HPC Workloads

Jacopo Marino, Fulvio Risso
Dept. of Control and Computer Engineering
Politecnico di Torino
Torino, Italy
{name.surname}@polito.it

Mauro Bighi
Dept. of Information Technology
PUNCH Torino S.p.A.
Torino, Italy
mauro.bighi@punchtorino.com

***Abstract*—The vast array of cloud providers present in today's market proffer a suite of High-Performance Computing (HPC) services. However, these offerings are characterized by significant variations in execution times and cost structures. Consequently, selecting the optimal cloud provider and configuring the features of the chosen computing instance (e.g. virtual machines) proves to be a challenging task for users intending to execute HPC workloads. This paper introduces a novel component designed for effortless integration with existing HPC scheduling systems. This module's primary function is to facilitate the selection of the most appropriate cloud provider for each distinct job, thereby empowering dynamic and adaptive cost-minimization strategies. Through the application of data augmentation techniques and the employment of Continuous Machine Learning, the system is endowed with the capability to operate efficiently with cloud providers that have not been previously utilized. Furthermore, it is capable of tracking the evolution of jobs over time. Our results show that this component can achieve consistent economic savings, based on the quality of the data used in the training phase.**

***Index Terms*—cloud computing, hpc, machine learning**

I. Introduction

The adoption of cloud computing technologies to run High-Performance Computing (HPC) workloads has steadily increased over the years. Users prepare models on their personal computers, upload them to the cloud and analyze the results when ready by either downloading the produced data or leveraging cloud-based remote desktops. Current HPC schedulers, both commercial products and open-source software, leverage several FIFO queues to schedule submitted workloads on a set of given cloud providers' instances (e.g., VMs). In this situation, the choice of the cloud provider and of the *type* and *number* of instances is left to the user. In fact, the scheduling simply follows the order of submissions of jobs, without considering their details or users' needs (such as minimizing cost or running time).

On the other side, different business constraints may exist. Some urgent tasks should be completed in the shortest time, paying less attention to the costs, while some deferrable workloads could be optimized to reduce costs at the expense of their duration. In this respect, notable differences exist between different cloud providers, as shown in Table I that reports an example of cost and running time of a real HPC task launched on four major cloud providers on Jan 2023.

TABLE I
Example of HPC workload costs in public cloud

Cloud provider	#Cores/VM	#VMs	Execution time	Cost
#1	36	4	10h 8m 54s	93.51€
#2	120	1	12h 16m 50s	51.77€
#3	30	4	17h 44m 12s	209.29€
#4	36	4	14h 9m 11s	104.73€

From this perspective, it becomes apparent that an assisting software solution is required to guide users in selecting the most efficient instance. This will facilitate the optimization of overall task costs by scheduling high-priority workloads on the fastest provider. Conversely, lower priority tasks can be allocated to the most cost-effective options to enable cost savings.

This paper proposes a novel component, that can be seamlessly integrated with existing HPC schedulers, that determines the best queue (hence, cloud provider) based on the selected optimization policy (i.e., duration vs cost), hence enabling dynamic and adaptive cost-optimization strategies. Given that costs and running time of an HPC job (e.g., Table I) are highly dependent on the workload itself, our component will leverage best-in-class machine-learning technologies, allowing the model to learn on the job with training on past data and on newer ones when available, hence improving its outcome continuously following step by step the evolution of the HPC tasks set.

The rest of this paper is structured as follows. Section II summarizes the current state of the art. Section III presents the architecture of the proposed solution, while the testing methodology and results are presented in Section IV. Finally, Section V concludes the paper and highlights possible future research directions.

II. Related work

The problem of dispatching HPC jobs has been addressed by several research papers, but also by several products, which leverage different heuristics to achieve load balancing across multiple providers, minimize the queue length and more.

Focusing on the scientific literature, Y.-K. Suh et al. [1] proposed a simulation runtime estimation scheme, *CLUTCH*, that achieved approximately a 14.2% growth in estimation

accuracy, compared to the state-of-the-art schemes at the time of writing. However, their solution does not address the possibility of scaling a job with multiple instances running in parallel, which is of fundamental importance for compute-intensive HPC tasks that need to span across multiple VMs. For instance, the same problem affects the next analyzed papers.

S. Sok et al. [2] proposed an approach called full rescheduling that allows online schedulers to move already running tasks from the public to the private cloud in order to save costs. Although this approach aims for a cost-optimal hybrid operation, the authors focused on handling the workload peaks using the public cloud, while this paper aims at dispatching workloads minimizing the cost or the running time, using on-premise or cloud instances. In addition, the authors do not consider the possibility of scaling on multiple instances.

B. Li et al. [3] formulated the task placement across multiple public clouds problem as an Integer Linear Programming (ILP), to minimize the cost of resources in a heterogeneous cloud environment, including multiple public clouds, while ensuring a proper level of QoS. However, the authors do not consider the possibility of scaling on multiple instances, just placing a task on a single VM.

Y. Balagoni et al. [4] proposed a hybrid cloud scheduler to address the problem of the estimation of application execution time in deadline-constrained applications running in hybrid cloud infrastructures. However, the proposed solution does not handle the case of scaling on multiple instances.

T.-P. Pham et al. [5] introduced a novel two-stage machine learning approach for predicting workflow task execution times for varying input data in the cloud. Although their approach relies on parameters reflecting runtime information and two stages of predictions, it does not consider the possibility of scaling on multiple instances.

F. Nadeem et al. [6] developed ML-based ensemble systems that employed three algorithms to balance their corresponding weaknesses and strengths, modeling the execution time of e-science workflows. Although these methods can also be applied effectively, without major modification, to other heterogeneous distributed environments such as the cloud, the authors focused on grid computing, which does not require such as low-latency interconnect between VMs.

N. J. Yadwadkar et al. [7] introduced PARIS, a data-driven system that efficiently selects suitable virtual machines (VMs) for workloads and user goals, resulting in a 45% cost reduction while maintaining performance. However, it does not consider the possibility to scale across multiple instances, limiting its applicability in High-Performance Computing (HPC) where parallel collaboration is essential.

M. Bilal et al. [8] introduced Vanir, an optimization framework for analytics clusters. It reduces search costs compared to other cloud optimizers. The proposed solution does not consider the close interconnection needed for running an HPC computation. Merely using multiple instances is not sufficient, as specific requirements must be met, and not all instances offered by cloud providers are suitable for this purpose.

Y. Wu et al. [9] introduced Vesta, a transfer learning approach for optimizing VM selection across multiple frameworks, achieving up to 51% performance improvement in experiments with Hadoop, Hive, and Spark. C. Chen-Chun et al. [10] proposed a deep neural network to predict Hadoop's job time based on historical execution data. C. Hsu et al. [11] proposed augmenting Bayesian Optimization with low-level performance information, which can reduce search cost and potentially improve performance. CherryPick, developed by A. Omid et al. [12], is a system utilizing Bayesian Optimization to automatically identify the most suitable cloud configuration for recurring big data analytics tasks. It is important to note that the authors of those papers focused on software/frameworks for big data analysis achieving good results, but however, they disregarded parallel machine computations commonly found in HPC clusters. The authors' focus on big data frameworks overlooks the need for horizontal scaling across multiple instances with the constraint of low-latency interconnection. This limitation makes the approaches less applicable to scenarios where multiple instances working in parallel are essential.

Kubecost [13] is a tool that allows users to see the spending associated with each allocated cloud-native Kubernetes resource, hence providing teams with transparent, accurate cost data reconciled with actual cloud bills. It has real-time alerting functionality and recurring reports that empower teams to take control of their Kubernetes-enabled infrastructure, stay within budgeted limits, and address monitoring interruptions immediately. However, it cannot be used to predict the best cloud provider for a future simulation.

This paper proposes a novel component for selecting the most suitable VM or set of them for HPC workloads while considering at the same time low-latency interconnections, cost optimization/ending-time constraints, and compatibility with existing commercial HPC schedulers.

III. Proposed Architecture

This Section presents the architecture of the prediction component, detailing its internal structure and its features.

A. Pre-runtime, runtime parameters, and output

Our predictor bases its results on the outcome of previous HPC jobs, leveraging two sets of parameters, namely pre-runtime parameters (PRPs) and runtime parameters (RPs), to predict the running time of future jobs. From the expected running time, we can easily derive the total job cost, since the cost associated with used resources (e.g., VMs) in each cloud provider is known.

PRPs are known a priori before the job is scheduled and are the following:

- **cloud**: it defines the cloud provider hosting the job (e.g., *oracle*, *azure*, *on-prem*, etc.), and it can assume a value in a predefined set.
- **software**: it defines the specific software used by the job and depends on the type of analysis that has to be done. For example, a possible value could be the *Abaqus*

software in the case of FEM (Finite Element Method) simulations.

- **cores**, **ram**, **vm_number** and **vm_type**: It defines the computing resources assigned to a node running on the given cloud provider, in terms of numbers of allocated CPU cores, memory, number and type of VMs used, which is provider-independent.

With respect to *vm_number* and *vm_type*, although cloud providers offer several types of VMs, HPC jobs require special capabilities such as a dedicated interface for high-speed data transfer (e.g., InfiniBand) to be executed efficiently. Hence, only a few types of VMs can be used, which are usually associated with a fixed amount of resources, i.e. CPU cores and RAM. Consequently, the four parameters *cores*, *ram*, *vm_number* and *vm_type* are closely related to each other and a value of the last two uniquely defines the values of the others.

Since the size of the available VMs may be different in each provider, the *vm_number* provides a way to allocate 'standard' VMs, whose size is similar across all cloud vendors. For example, *vm_number = 1* could correspond to a VM with 36 cores and 512GB RAM on Cloud #1, while it results in a VM with 32 cores and 448GB RAM on Cloud #2.

RPs, listed below, are parameters that depend on the job execution, hence they can be only known when the job terminates:

- **cput**: time spent by the CPU to execute the job.
- **ncpus**: number of vCPUs used by a job (it could happen that not all cores are used).
- **memory**: peak memory used by the job.
- **vmem**: peak virtual memory used by the job, which includes also the memory consumed by additional components such as memory-mapped files and swap space.

RPs are used for the prediction of the running time, but they can also be used to understand if the instance selected by the user would be fully utilized. The idea is to leverage the *ncpus* to predict how jobs will use the instance and in case of under-utilization (defined by a proper threshold), the infrastructure manager could choose for future jobs another *vm_number* that is more appropriate, avoiding a waste of resources and reducing costs.

B. Two-stage prediction component

The prediction component, which is considered a black box by the scheduler, consists of two sub-predictors, both with the same structure, operating based on the output of the ML algorithms that will be shown in Section III-E.

The first sub-predictor leverages PRPs to predict RPs, while the second uses all known data (PRPs and RPs) to predict the running time of the job, as shown in Figure 1. In fact, the output of the system is a tuple composed of the predicted running time required by the job to complete, and the related platform that should be used (a cloud provider or on-premise) to obtain this time, plus the estimated cost of this solution $(Cloud, T_{pred}, C_{pred})$. The latter is obtained by multiplying the hourly cost of the chosen *vm_type* by the predicted running time and the *vm_number*. The system can be used to determine the cheapest or the fastest cloud provider, changing an internal parameter.

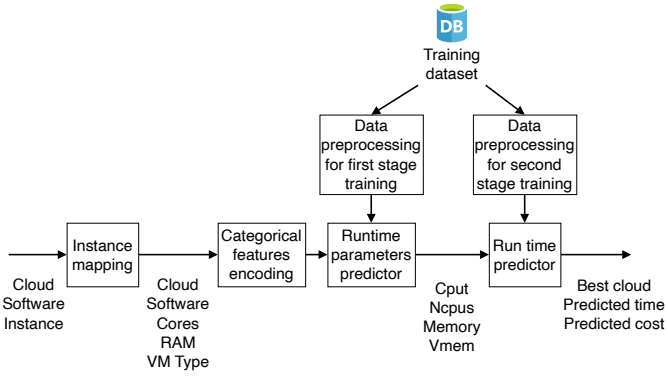

Fig. 1. System workflow

A predictor can be seen as a block that implements a function f, getting $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ as input and y as output, where $\mathbf{x}$ is the input set and y the predicted running time of a job with the given parameters. The function f serves as a mapping from a given input, x, to a corresponding output, y, which is a continuous positive value. The function f is determined by the data used in the training, and it is a machine learning algorithm for regression tasks.

When a user would like to run a job on a specific cloud provider among the N available, the system makes a prediction for all the providers, using the same user inputs, changing only the provider-related ones. Predictions are made by using the ML model marked as the best one. The results are then compared and returned to the user, highlighting the cheapest and the fastest options.

In the corporate world, job cost and running time often conflict with each other, and it is the users who ultimately determine the approach to be used based on predictions of the above values. Whether it's reducing costs or meeting deadlines sooner, users make the decision and determine how much money should be allocated based on the urgency of the task at hand. Therefore, highly advanced decision-making systems are not necessary since the users themselves assume the responsibility of selecting the appropriate course of action.

C. Training set

Data used to train the ML algorithms is derived from the logs collected from previous HPC jobs. The first step is to pre-process raw data from existing logs to extract the parameters for the training set, i.e., PRPs and RPs as presented in Section III-A. The second step maps categorical inputs into numerical ones using a mapping function. It converts string values into numbers suitable for the ML algorithms, i.e. the *azure* string can become a number such as 3, or it can become an array of boolean values such as in the One-Hot Encoding. Both strategies are suitable for the purpose of this paper; the use of one or the other is an implementation choice.

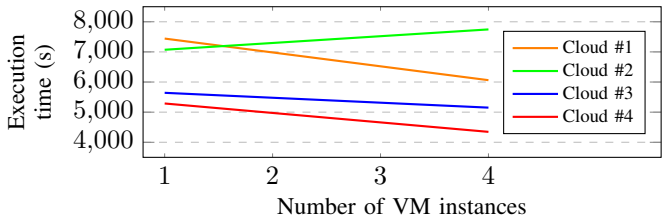

Fig. 2. Instances' scaling scenario, each line represents a cloud provider.

The training set (from this point forward referred to only as *dataset*) contains an entry for each job previously submitted by users, grouped by *projects*, which refers to a given type of simulation and/or product. For example, the simulation of a diesel engine for marine usage may have very different characteristics from a simulation of a gasoline engine for vehicles. Grouping the data set by *projects* allows a reduction of noise in predictors, hence enabling more accurate results.

D. Data augmentation

The accuracy of the prediction obtained by ML algorithms greatly depends on the availability of large training datasets. However, particularly in case of new projects in which a few jobs have been completed, the cardinality of the training set is limited, with a potential huge impact on the accuracy of the predictor. This problem is further exacerbated by grouping the data per project, with some projects with hundreds of samples and others with only a few.

Our solution leverages two data augmentation algorithms (each one associated to one sub-predictor) to reach about 10K examples per project, both based on the same two steps.

Step 1. Each entry is replicated n times, depending on the number of cloud providers active and usable, changing the *cloud* value and the related ones, i.e., *core*, *ram*, *ncpus*, *cput* and *runtime*. The *vm_type* and *vm_number* values are left unchanged because the goal is not to scale vertically between the same provider using a different configuration, but to move to different providers. This change is achieved by using the scalability curves shown in Figure 2, which have been derived from existing experience (i.e., jobs) to determine to what extent a job can improve (or worsen, in case of *Cloud #2*) its running time when adding new VMs in parallel. This enables our system to map the execution time of a job completed on one cloud provider, compared to its potential execution on another provider. Given one training example and the cloud to map on, parameters are scaled and the newly crafted example is added to the augmented training set.

Estimations between different cloud providers are done by keeping the same *vm_number*: each number is associated with a score, which is the running time on that specific platform with that specific resource. It is possible to estimate the performance of a job on other cloud providers, by simply taking the ratio of the new and the old scores, and multiplying by it the execution time: the result is the estimation of the time on the new cloud provider chosen.

Step 2. Each entry is replicated j times, multiplying the RPs and the running time for a coefficient c that relies on a randomly generated value Δ, which follows a linear distribution characterized by a small range. This distribution ensures that Δ remains within a magnitude of less than 5%. The computation for this coefficient is as follows:

$$c = 1 + random(-\Delta, \Delta), \quad 0 \leq \Delta \leq 0.05 \tag{1}$$

The number of augmented samples j is derived from the total number of target samples for the ML predictor to work, i.e., about 10K per cloud provider within a given project.

E. Machine Learning predictors

The two sub-predictors have the same structure (hence, similar training, validation, and prediction phases) while differing only in their $(inputs, output)$ tuple. Machine learning algorithms can be fine-tuned by setting the hyperparameters to better fit the training data. Each predictor comprises various ML algorithms, all of which are trained during the training phase. The first step is to train the ML algorithms by changing the hyperparameters by orders of magnitude to find suitable ranges for grid search. These ranges are specific for each hyperparameter of each algorithm, e.g. Multi-Layer Perceptron, Random Forest, K-Nearest Neighbors, etc. The second step uses these ranges for the grid search that will be run by the system to find the best hyperparameters of the ML algorithms. Once the search finishes, the final training is done and the model is ready for predictions.

The validation of the ML algorithm that is being selected as a predictor within each specific project is done using a portion of the initial dataset, splitting it into training and validation sets, with a ratio of 80/20. The metric chosen for the selection of the best ML model is the Mean Absolute Percentage Error (MAPE): the algorithm with the lowest value is marked as the *best* one and used for predictions.

F. Continuous Machine Learning (CML)

In order to improve the accuracy of the predictors, we leverage Continuous Machine Learning (CML) techniques to periodically re-train the ML models to include the outcome of the recent jobs. The idea is to re-train models after n new run of jobs, to follow the real scenario of data and improve the quality of our predictions.

An ML model contains all the information needed to obtain the output from the inputs, that is, all the internal parameters that define the ML algorithm. After re-training, the chosen hyperparameters may be the same, but models are different because the training set has changed as new examples have been added. The training phase is done in the same way as Section III-C; hence, after each re-training, the selected ML algorithm may change, as we choose the one that is more appropriate for the current set of past executions.

IV. Results

Our algorithms have been validated by leveraging data available from a company that designs and builds automotive

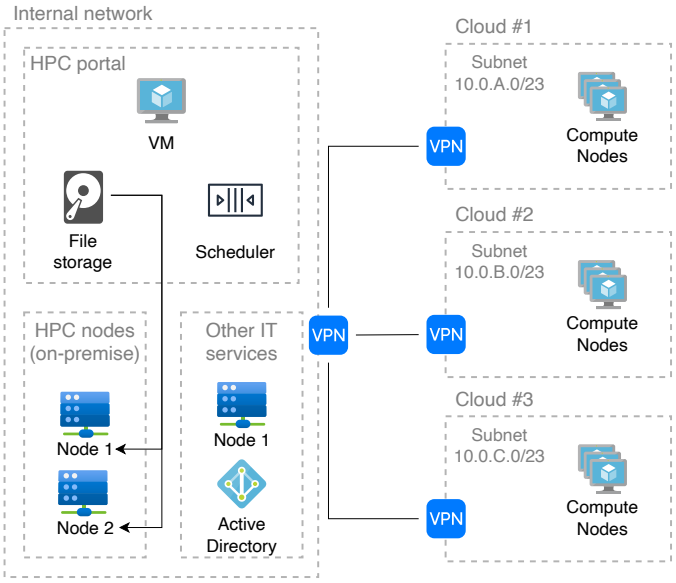

Fig. 3. High-level view of the HPC infrastructure used in the validation.

engines. As part of their daily work, analysts need to run HPC jobs to define the draft model of the engine (or the sub-engine part) and then fine-tune all parameters of the product.

A. Experimental Environment

The jobs run by analysts are HPC workloads executed on the available infrastructure, which is a hybrid cloud made by on-premise computational nodes alongside on-demand instances deployed on different cloud providers, as shown in Figure 3. In this particular case, the cost would always be minimized by running the jobs on-premise, but the execution time may be minimized when running the jobs on a public cloud infrastructure.

HPC jobs are handled by a commercial scheduler, which implements a simple FIFO policy to dispatch the jobs on either one of the available cloud providers or on on-premise nodes. Each queue is associated with a scenario, i.e., a combination of the cloud provider and the *vm_number* to be used by the job. The specific scenario to be used is chosen by the user who starts the HPC workload, then jobs on the same queue are dispatched with the FIFO policy.

Users decide the configuration they wish to use for jobs, so the scheduler only takes care of their submission. It only deals with deployment constraints, such as the maximum number of instances active at the same time and the creation of instances on-premises or on providers. It puts jobs on hold if the maximum number of instances has been reached, then proceeds to schedule them when the previous ones finish, collecting logs. It is always active waiting for new jobs when the queues are empty.

B. Test conditions

In order to validate the contribution of the different features introduced in Section III, such as **data augmentation** and

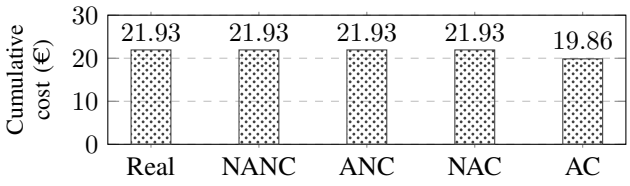

Fig. 4. Project A, real vs. estimated cost.

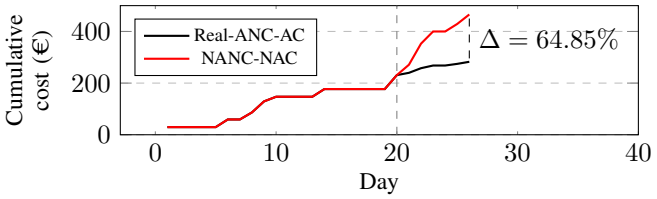

Fig. 5. Project B, real vs. estimated cost.

CML, we identify our results with the following labels: *NANC*, *ANC*, *NAC* and *AC*, where *A* stands for augmentation, *C* for CML, and the *N* prefix means *not*.

Data comes from two real projects (*Project A* and *Project B*) which have been used for both training and validation, and eventually to derive the run-time predictions of the system. FEM simulation is a numerical technique that approximates complex mathematical problems by subdividing them into simpler, smaller parts called finite elements. By solving these elements individually and combining the results, it can accurately model complex behaviors or systems. The simulations of Project A involve multiple iterations of the same model, each with distinct parameters, tailored to refine a distinct facet of a specific engine. On the other hand, The simulations of Project B are dedicated to examining the operational characteristics of a different engine variant, typically exhibiting extended execution duration. Both leverage the power of parallel computing, meaning each participating instance plays a crucial role in generating the overall output of the simulation. The data used to test the system and collect the results has never been seen before by the system. The test set must not be used during the training phase, otherwise, the system could overfit it instead of being able to generalize on new data.

The test set implements a loop in a way to pass to the system the data requires to obtain n predictions, where n is the number of available cloud providers. Predictions are then used to identify the cheapest provider. If the latter is the same used by the real job from the test set, then the running time of the latter is used to compute the cost. Otherwise, if the providers differ, the curves presented in the Section III-D are used to estimate the running time of the job on the predicted provider, i.e., the estimated cost (and time) on the provider chosen by the prediction system.

In the end, we can compare the real cost of the HPC job with its estimated cost, having a better way to compare the four available providers.

C. Results analysis

Data augmentation provides the system with examples that could not be present because users might have used only some

providers. Both Figure 4 and Figure 5 show cumulative results, but the first is presented as bars because the time interval of jobs is two days, while the second shows results over a wider time window of 26 days.

Data augmentation has been shown to enhance the performance of the system, although its impact may vary depending on the characteristics of the tasks and the number of examples. Specifically, in Project A, where the HPC tasks demonstrate high similarities and numerous examples are present, the influence of data augmentation on system accuracy is minimal. Conversely, for Project B, characterized by more significant task variance and a smaller set of examples, data augmentation proves crucial in aiding the system to generalize through an expanded pool of training examples. This divergence in the utility of data augmentation is illustrated by the *NANC* and *ANC* bars of Figure 4, and the *NANC-NAC* and *Real-ANC-AC* lines of Figure 5. One key instance from Project B highlights the need for data augmentation: on day 20, users changed the *vm_number* from 4 to 3. The dataset contained some instances of *vm_number* being 3, albeit for a different cloud provider. Due to the absence of this information and lack of data augmentation, the system predicted cost savings with a different provider, which eventually proved incorrect, leading to a cost increase of up to 64.85%.

Without data augmentation, the system's MAPE was lower. However, considering a broader predictive time frame led to better cloud provider predictions, even though it increased the error. This demonstrates that avoiding overfitting and maintaining an error threshold is crucial for machine learning algorithms to accurately predict unseen examples.

Continuous Machine Learning (CML) is particularly beneficial in scenarios with a large influx of new simulations post-training, as observed in Project A, resulting in a cost reduction of about 9.46%, as represented by the *AC* bar in Figure 4. However, in situations like Project B, with far fewer new examples, CML seems to exert negligible influence on the quality of the solution, as depicted by the *NANC-NAC* line in Figure 5.

Based on the collected results, the system performs satisfactorily using an initial single training phase. However, performance could vary if the initial training set quality is poor. This can be mitigated by implementing data augmentation and/or CML, thus bridging any knowledge gaps until newer examples become available.

V. Conclusions

This paper presents a novel pre-processing component, equipped to accurately predict the execution time of HPC jobs. The module harnesses historical data to continuously refine its predictive model, thereby enabling either the optimization of the execution time or the economic cost, based on the selection of the most suitable infrastructure provider. The precision of the predictive algorithm is best when leveraging high-quality data, as demonstrated by Project A. Here, the system, when configured to choose the most cost-effective cloud provider, recorded approximately 9.46% in cost savings against the baseline in the test environment.

However, the study underscores that the algorithm may yield inaccurate predictions when reliant on low-quality data, as exemplified by Project B, where errors could have escalated costs by up to 64.85%. To counter such inaccuracies, the system necessitates enhancements in data quality via data augmentation and CML, which ensures significant improvement in precision over time. In real-world environments, predictive precision is evaluated by comparing predicted execution times against actual outcomes, alerting users to any deviations from optimal performance and facilitating system re-training when required. Despite potential challenges, the overriding advantages of data augmentation and CML, especially with a substantial volume of structured data, position ML systems to outperform human decision-making processes, underscoring the compelling benefits of the proposed component.

References

[1] Y.-K. Suh, S. Kim, and J. Kim, "Clutch: A clustering-driven runtime estimation scheme for scientific simulations," *IEEE Access*, vol. 8, pp. 220 710–220 722, 2020.

[2] S. Sok, C. Plewnia, S. Tanachutiwat, and H. Lichter, "Optimization of compute costs in hybrid clouds with full rescheduling," in *2020 IEEE International Conference on Smart Cloud (SmartCloud)*, 2020, pp. 35–40.

[3] B. Li, Z. Zhao, Y. Guan, N. Ai, X. Dong, and B. Wu, "Task placement across multiple public clouds with deadline constraints for smart factory," *IEEE Access*, vol. 6, pp. 1560–1564, 2018.

[4] Y. Balagoni and R. R. Rao, "A cost-effective sla-aware scheduling for hybrid cloud environment," in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2016, pp. 1–7.

[5] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 256–268, 2020.

[6] F. Nadeem, D. Alghazzawi, A. Mashat, K. Faqeeh, and A. Almalaise, "Using machine learning ensemble methods to predict execution time of e-science workflows in heterogeneous distributed systems," *IEEE Access*, vol. 7, pp. 25 138–25 149, 2019.

[7] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best vm across multiple public clouds: A data-driven performance modeling approach," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 452–465.

[8] M. Bilal, M. Canini, and R. Rodrigues, "Finding the right cloud configuration for analytics clusters," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 208–222.

[9] Y. Wu, H. Wu, Y. Xu, Y. Hu, W. Zhang, H. Zhong, and T. Huang, "Best vm selection for big data applications across multiple frameworks by transfer learning," in *Proceedings of the 50th International Conference on Parallel Processing*, ser. ICPP '21. New York, NY, USA: Association for Computing Machinery, 2021.

[10] C.-C. Chen, K.-S. Wang, Y.-T. Hsiao, and J. Chou, "Albert: An automatic learning based execution and resource management system for optimizing hadoop workload in clouds," *Journal of Parallel and Distributed Computing*, vol. 168, pp. 45–56, 2022.

[11] C.-J. Hsu, V. Nair, V. W. Freeh, and T. Menzies, "Arrow: Low-level augmented bayesian optimization for finding the best cloud vm," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 660–670.

[12] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 469–482.

[13] Stackwatch, Inc. (2023) Kubecost. [Online]. Available: https://www.kubecost.com