Machine Learning Regression-based Prediction for Improving Performance and Energy Consumption in HPC platforms

Micaella Coelho[1], Kary Ocaña[1], André Pereira[2], Alexandre Porto[1], Douglas O. Cardoso[3], Arthur Lorenzon[4], Rui Oliveira[2], Philippe O. A. Navaux[4], Carla Osthoff[1]

[1] National Laboratory of Scientific Computing, LNCC, Rio de Janeiro, Brazil
{micaella,xandao,karyann,osthoff}@lncc.br
[2] University of Minho & HASLab INESC TEC, Campus Gualtar, Braga, Portugal
{andre.martins.pereira,rui.oliveira}@inesctec.pt
[3] Centre for Linguistics, Faculty of Arts and Humanities, University of Porto, Porto, Portugal
docardoso@letras.up.pt
[4] Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
{navaux,arthur.lorenzon}@inf.ufrgs.br

Abstract. High-performance computing is pivotal for processing large datasets and executing complex simulations, ensuring faster and more accurate results. Improving the performance of software and scientific workflows in such environments requires careful analysis of their computational behavior and energy consumption. Therefore, maximizing computational throughput in these environments, through adequate software configuration and resource allocation, is essential for improving performance. The work presented in this paper focuses on leveraging regression-based machine learning and decision trees to analyze and optimize resource allocation in high-performance computing environments based on application's performance and energy metrics. Applied to a bioinformatics case study, these models enable informed decision-making by selecting the appropriate computing resources to enhance the performance of a phylogenomics software. Our contribution is to better explore and understand the efficient resource management of supercomputers, namely Santos Dumont. We show that the predictions for application's execution time using the proposed method are accurate for various amounts of computing nodes, while energy consumption predictions are less precise. The application parameters most relevant for this work are identified and the relative importance of each application parameter to the accuracy of the prediction is analysed.

Keywords: Machine learning · High-performance computing · Scientific applications · Bioinformatics · Resource management.

1 Introduction

High-performance computing (HPC) resources are pivotal for processing large-scale datasets and conducting intricate software simulations in several fields, such as bioinformatics, physics, and chemistry. Adequately allocating and efficiently using these resources is important to achieve optimal performance while controlling energy consumption. As HPC systems handle increasingly complex workloads, minimizing energy usage while maximizing computational throughput becomes essential, as data centers account for 2% of the global energy consumption as of 2020, and is expected to increase to 10% by 2030[5].

Machine learning (ML) techniques, such as regression-based prediction and decision trees, are employed to analyze logs generated from the execution of scientific software to predict their computational behavior on different resources. Such predictions, if accurate, are key to allocating the adequate type and amount of computational resources to ensure efficient execution of that software. We propose an initial regression ML model, which configures application's execution and predicts the best computational resources to improve its execution time and energy efficiency for HPC environments.

The bioinformatics tool RAxML 8.2.12 [3] is used as case study, executing on the HPC resources of the Santos Dumont supercomputer (SDumont) [6]. The aim is to optimize resource allocation for bioinformatics workloads using the proposed regression model, based on decision trees, thereby improving performance and reducing energy consumption. By leveraging data from past executions and system parameters, informed decision-making is enabled when configuring RAxML executions. These predictive capabilities enhance phylogenomics application performance and ensure efficient HPC resource management.

The proposed ML approach uses logs from previous RAxML software executions to identify the most relevant parameters for the decision-making process. This includes application information such as the number of threads, nodes, and dataset sizes [5] [8]. The study explores how these optimizations improve efficiency and conserve energy, establishing regression-based prediction as pivotal in advancing HPC resource allocation. This approach can potentially be applied to other life sciences fields, such as computational biology and molecular evolution, on advanced infrastructures like SDumont.

Considering the aforementioned scenario, this study evaluates regression model predictors using real data from scientific application executions. Our contributions include: (*i*) identifying key parameters — related to software, computational architecture, and datasets - for predicting execution time and energy consumption in bioinformatics applications using machine learning; (*ii*) demonstrating the impact of the important input variables on predictor accuracy; and (*iii*) assessing predictor accuracy in a real bioinformatics scenario on a production HPC system.

[5] https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks
[6] https://sdumont.lncc.br/

This paper is organized as follows. In Section 2, related works are presented, in Section 3 we recall the definitions of regression predictors and bioinformatics. Section 4 describes the prediction ML models and the data used for our experiments, and how we have recovered the needed data that was not directly available. Section 5 presents the numerical results and Section 6 provides the conclusion and final remarks.

2 Related Work

This work explores the impact of different configurations of the bioinformatics software RAxML on its performance and scalability in Santos Dumont. A machine learning (ML) approach is utilized to choose the best configuration and adequate resource allocation in supercomputers. Features such as the node type, a number of cores, input data size, and RAxML performance measurements were used as training data for the ML model, similarly to other work in this field.

In [19], a performance analysis of the parallel versions implemented in RAxML, supporting the Hybrid version as the most efficient, is presented. [20] present a comparative analysis between the PhyML, IQ-TREE and RAxML/ExaML programs, concluding that RAxML, in addition to being more scalable, generates better-quality tree topologies. This paper explores the performance of RAxML in the SDumont supercomputer, exploring environment configurations and RAxML settings (as bootstrap values and data size features) that influence executions. They are complementary work since exploring HPC software as RAxML presents several challenges, such as coupling to HPC infrastructures to demonstrate performance behavior and scalability for processing parallel and distributed executions.

The adequate allocation of computing resources to scientific software has been extensively studied. The authors of [11] propose an ML model that can deal with a large set of attributes related to both the work being performed and the HPC system. An extension to the Predicting Query Runtime algorithm is also proposed for the regression problem, which chooses the best learning model for the data related to the leaves of the tree. With this method, the authors could predict execution times, memory and disk usage for BLAST and RAxML applications, the latter also evaluated in our study. However, the reliance on various specialized models with varying accuracies indicates that the effectiveness of the proposed approach can vary based on the characteristics of the data and the application.

The authors of [12] evaluate multiple linear regression and neural network models for building performance models to predict tasks' execution time, similar to our work. As in other studies, metrics were collected during the execution of the work and were then used to train the models. The difference in the authors' proposal is that ML is used in two stages. The first stage comprises a trained model with the collected metrics, which is then used to predict a subset of the most relevant metrics. In the second step, the most relevant metrics are used to build the performance model to predict task execution times effectively. However,

while improving prediction accuracy for specific cases, the two-step ML approach and the extensive feature selection process introduce significant complexity and computational overhead.

The authors of [13] use supervised and unsupervised machine learning techniques to predict the waiting times of jobs in the execution queues of HPC management systems. The data used as input to the models was collected from a history of real jobs performed in the system. Correct management of execution queues is critical for the efficient use of system resources and for an adequate waiting time during which tasks wait for their turn to be executed. To achieve this objective, two approaches were used: a classification model to predict the time intervals in the queues and a regression model to predict the effective waiting time for each job to be executed. However, the model accuracy was inconsistent across all queues, even with a model specialized for each queue type. This inconsistency indicates that queue-specific models may not always offer a clear advantage over a unified model that handles all queues collectively.

The authors of [14] evaluate several machine learning techniques to predict the execution times of jobs in queues of HPC systems. The work proposes and evaluates a two-stage prediction method, with the main objective of assessing the accuracy of various types of models: general models based on the flow of work submitted to the system, or specialized models based on the characteristics of each job. The authors point out that certain ML techniques did not consistently outperform others and may struggle to handle dynamic changes in workload patterns over time. The temporal variability in the data can impact model accuracy, indicating a need for more robust methods to adapt to changing conditions in HPC environments.

The authors of [15] focus on evaluating the performance and resource allocation for MPI applications on HPC systems. To this end, low-level information, such as branch counters, loops, and communications between MPI processes, are collected through the automatic instrumentation of the application. Using this data, an ML model based on Random Forests is trained with the collected data and then used to predict the execution time of an MPI application for a new, unused input. However, in environments where low-level application data is not available or difficult to collect, the effectiveness of the models is compromised.

3 Background

3.1 Machine Learning for time and energy prediction

Machine learning regression-based prediction models can play an important role in optimizing performance and energy consumption in HPC environments. These models can predict and manage energy consumption in HPC workloads, aiming to reduce energy use in supercomputers without compromising the performance of software and resource and task management systems [9]. They utilize statistical algorithms to predict outcomes based on historical data—for example, forecasting computational time and energy usage of applications based on prior

resource allocations. In HPC environments, where efficient resource management is crucial, regression models provide accurate forecasts of performance metrics, facilitating more effective energy management.

Regression Model Formulation

The Extra Trees Regressor (ETR) [1], a machine learning technique, enhances traditional decision tree algorithms by introducing randomness in feature selection and split points. This randomness reduces variance and improves the model's generalization ability to unseen data, enhancing predictive accuracy and reliability in complex HPC environments. This model was chosen due to the advantage of not requiring fine-tuning of settings to achieve competitive performance compared to other models with the same purpose. Additionally, it provides an estimate during the training of the degree of influence of each input variable on predicting the output variable, which is useful for assessing the importance of input parameters in predicting output variables.

The ETR predicts the target variable y based on features X. The model fits multiple decision trees on various subsets of the dataset and averages their predictions to improve robustness and reduce overfitting.

Cross-Validation

Resampling methods such as cross-validation can be employed to evaluate the performance of the ETR. This technique involves training and evaluating the model on distinct datasets by dividing the original sample into training and testing sets. The k-fold cross-validation method splits the dataset into k subsets of similar size. One subset is used as the test set, while the remaining k-1 subsets are used to train the model. After training, the model is evaluated using the test set and an evaluation metric, such as the Mean Absolute Error (MAE). This process is repeated n times, with new formations of the subsets in each iteration, ensuring a robust evaluation of the model's performance. At the end of the iterations, the values of the evaluation metrics obtained in each iteration are used to calculate the average, providing a stable and reliable estimate of the model's performance across different data subsets [10].

The Mean Absolute Error (MAE) [7] is a metric used in regression problems to evaluate a model's accuracy. The MAE is calculated by summing the absolute differences between the predictions and the actual values, divided by the total number of observations. The lower the MAE value, the better the model's performance, indicating that the predictions are closer to the actual values. It can be calculated as follows:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

where n is the number of samples, y_i is the true value, and $\hat{y}_i$ is the predicted value for the i-th sample.

3.2 RAxML for parallelized phylogenetic analysis on multi-core architectures

RAxML (Randomized Axelerated Maximum Likelihood) [3] is a leading software for phylogenetic analysis, using advanced maximum likelihood algorithms. Its standout feature is a rapid tree search algorithm that consistently yields trees with optimal likelihood scores, supported by robust parallelization capabilities across various computing environments. RAxML offers fine-grain parallelization using PThreads for multi-core systems and coarse-grain parallelization via MPI [16] for independent tree searches. The hybrid MPI/PThreads version combines these for enhanced efficiency. Sequential mode suits small to medium datasets in phylogenetic experiments, while parallel versions (MPI, PThreads, or Hybrid) optimize large dataset processing in phylogenomics [3].

RAxML utilizes SSE3, AVX, and AVX2 instructions so that vectorization improves the performance of likelihood and parsimony computations on CPU resources. PThreads' efficiency in parallelization depends on genome alignment length, performing optimally with longer alignments but strongly influenced by hardware specifics. The RAxML computational efficiency is further impacted by data complexity; datasets with more states (e.g., 4 for DNA, 20 for amino acids) require fewer site patterns per thread/core for effective parallel execution. Moreover, computational demands vary between models; the GAMMA [17] model necessitates more computations compared to the CAT model, which requires approximately one-fourth the computational resources of GAMMA [8].

4 Computational Time and Energy Prediction of the ML Model

4.1 The SDumont Supercomputer

We performed an empirical study using data from a real bioinformatics database provided by the LNCC Bioinformatics and SENAPAD groups, utilizing the Santos Dumont (SDumont) supercomputer located in Rio de Janeiro, Brazil. This system offers a processing capability of 5.1 Petaflops and consists of 36,472 CPU cores distributed across 1,134 computational nodes, which are interconnected by a high-speed, low-latency Infiniband network. The experiments were conducted on SDumont using base nodes equipped with two Intel Xeon E5-2695v2 CPUs (24 cores, 64 GB RAM). Job management was handled by the Slurm Workload Manager (v17.02) [6]. For more detailed information about SDumont, please visit [7]. The bioinformatics application utilized in our study is the phylogenomics software RAxML 8.2.12, compiled with AVX to enhance computational efficiency, and operated in MPI/PThread mode [3].

[7] https://sdumont.lncc.br/

4.2 The Dataset Aquisition for ML

The dataset for training the ML model was obtained from multiple runs of RAxML, capturing data on runtime, memory usage, and energy consumption while varying the input file sizes, bootstrap values, and computational settings at SDumont, including threads and nodes. Although trained with data from HPC environments, the training takes place outside these environments, aiming to optimize resource allocation without adding overhead to job processing. The bootstrap method, used in phylogenetic studies to create multiple simulated samples, helps build more accurate phylogenetic trees. Although increasing the number of bootstrap replicas enhances the phylogenetic analysis, it also increases the demand for processing time and computational resources. We used fifteen genomes of the Dengue virus (DENV) from four serotypes (DENV 1-4), originating from countries like Mexico, USA, Venezuela, Peru, Colombia, and Brazil, ranging from 98,484 to 1,158,854 bytes with alignment lengths between 10,782 and 11,145. Details about the genomes are available in the Table 1. Variations in *bootstrap* parameters were 10, 100, 1,000, and 2,000, and the number of SDumont *nodes* used varied between 1, 5, and 10, and *threads* per node ranged from 2, 4, 8, 12, to 24 to accommodate different processing demands.

RAxML behaviour was logged throughout January 22 to March 31, 2024, included five iterations of RAxML executions in experiments, with computational data collected using Slurm's sacct [18] command. With sacct we gathered most important metrics for a preliminary study, such as CPU time, allocated memory, and energy consumption, all of which were stored in a CSV file and used as input variables for creating the training database for the ML model.

Table 1. Features for Fifteen Dengue Virus (DENV) Genome Serotypes

Serotype	**Country**	**Genomes**	**Size (bytes)**	**Length (nt)**
1	Mexico	105	1,158,854	10,911
1	SouthAm	70	772,474	10,925
1	Nicaragua	69	766,479	10,973
1	Brazil	43	484,520	11,145
1	USA	34	377,019	10,949
1	Colombia	17	187,026	10,881
2	Venezuela	46	513,885	11,043
2	Peru	26	287,037	10,903
2	Mexico	22	242,689	10,910
2	Colombia	18	198,574	10,904
3	Peru	48	528,982	10,880
3	Colombia	21	231,143	10,887
4	Brazil	61	675,852	10,945
4	USA	25	272,549	10,782
4	Colombia	9	98,484	10,826

4.3 Identifying the Important Input Variables

After reconstructing the detailed information about application behaviour, we identified and included seven candidate output variables that are most relevant for predicting execution time, energy consumption, and memory usage. Table 2 presents a summary of these seven output variables.

We applied supervised machine learning to predict time, memory usage, and energy consumption based on input variables. The ETR model from the Scikit-Learn library [2] version 1.2.2 was chosen for its ability to provide accurate estimates for output numerical variables without the need for fine-tuning, and also for elucidating the importance of input variables. ETR operates with multiple decision trees that work independently, and their average predictions tend to be more accurate and robust than those of a single tree. Each tree in the model develops decision rules from input variables, helping to understand how they influence the outcomes. We used ETR with its default configuration, consisting of 100 trees and unlimited depth, allowing the trees to grow until the leaves are pure or contain fewer than two samples.

Table 2. Output Variables in Model Prediction Analysis

Output variables	**Information**
MaxVMSize	Maximum virtual memory size allocated.
AveVMSize	Average virtual memory size used by the job if paging or swap was utilized.
MaxRSS	Maximum RAM memory size allocated.
AveCPU	Average CPU time, which is the average time multiplied by the number of available cores.
CPUTime	Total CPU time, which is the total time of one core multiplied by the total number of cores available.
Elapsed	Total job duration.
ConsumedEnergyRaw	Total energy consumed by all tasks in a job, in joules.

We utilized this algorithm on the dataset to predict output variables such as time, memory, and energy consumption, while identifying the input variables for prediction. As an illustration of the results, Fig. 1 presents the importance scores of these input variables. The figure shows boxplots of scores for all attributes of the model (input variables), displaying minimum, median, mean, and maximum scores. Each input variable is represented by a colored boxplot: green for number of *Node*, orange for number of *Thread*, blue for *Bootstrap*, and pink for input dataset *Size*.

The input variables, listed in descending order of importance based on the average scores (green triangles), are: *Size*, *Bootstrap*, *Thread* and *Node*. The analysis reveals that the *Node* variable exhibits the least dispersion, indicated by a narrow interquartile range, suggesting low uncertainty in its importance. In contrast, the *Thread* and *Size* variables show significant interquartile ranges, suggesting that their importance can vary widely depending on the predicted

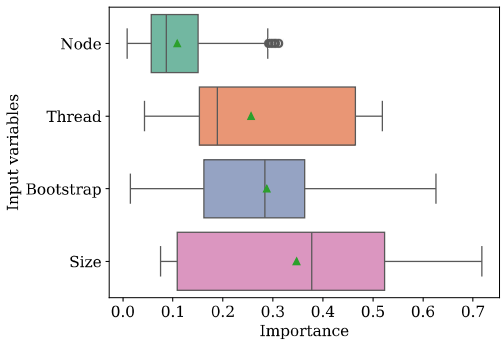


Fig. 1. Boxplot of score of the input variable importance.

output variable. This may be expected since the *threads* and dataset *size* an application uses doesn't necessarily mean it is compute intensive, which woreld translate into longer execution times and higher energy consumption. Thus, it is expected that there may be a low correlation between these variable and the time and energy that the model attempts to predict.

Addressing the previous query, Fig. 2 provides a detailed view of the average relative importance, allowing for the assessment of each input variable's importance to the predicted output variable. The figure displays importance values on the X-axis, output variables on the Y-axis, and uses bars to represent the importance of each input variable: blue for *Bootstrap*, green for *Node*, pink for *Size* and orange for *Thread*.

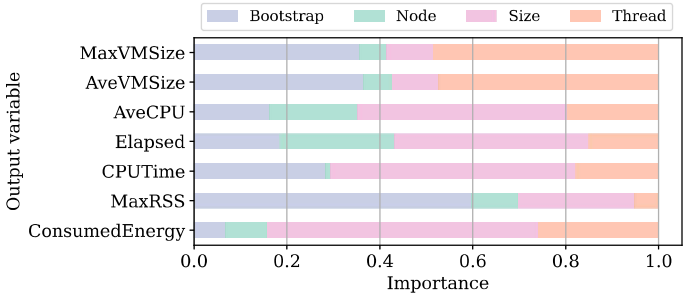


Fig. 2. Importance scores of input variables by output variable.

It was observed that Thread emerges as the most influential input variable in the regression for *MaxVMSize* and *AveVMSize* output variables, whereas *Node* shows the least importance. Conversely, for output variables related to computational time and energy (*AveCPU*, *Elapsed*, *CPUTime*, and *ConsumedEnergy*), genome *Size* stands out as the most significant input variable. In the case of *MaxRSS*, *Bootstrap* is identified as the most important input variable. Interest-

ingly, the importance of Size and Thread as input variables varies depending on the specific output variable, which affects predictions.

5 Computational Time and Energy Assessment of the ML Model

5.1 Model Output Accuracy

The regression analysis employed k-fold cross-validation and the MAE metric to evaluate the model's performance, providing insights into its accuracy across various aspects of the data. For each output variable, we conducted 100 rounds of 2-fold cross-validation, with each round randomly splitting the data into 50% for training and 50% for testing, resulting in 200 MAE measurements for the ETR model. Additionally, each MAE calculation included an analysis of the importance of input variables, estimated during the training process from this dataset. To ensure comparability across different scales, we normalized all output variables for the MAE calculations presented in Fig. 3, but retained the original scales in subsequent figures for concrete predictions.

Fig. 3 depicts a boxplot of MAE values comparing predicted and database values for time, memory, and energy using the ETR. The machine learning approach showed good results for the output variables *AveCPU*, *Elapsed*, and *CPUTime*, exhibiting low average MAE values and consistency across 100 folds. This may be due to the strong correlation between the mentioned input and output variables, as well as the ability of the ExtraTrees model in capturing these relationships more accurately than other output variables.

In contrast, for the memory-related output variables *MaxVMSize*, *AveVMSize*, and *MaxRSS*, as well as for the *ConsumedEnergy* variable, the model exhibited higher average MAE values and greater variability in the boxplots for *MaxRSS* and *ConsumedEnergy*, indicating variable performance across different folds. It can be concluded that the model is not as effective in predicting energy consumption as it is in estimating execution time, possibly due to a weak correlation between the input and output variables or the complexity of these relationships being difficult to capture accurately with the provided input variables.

5.2 Regression Model Performance Assessment

The analysis of Figs. 4 and 5 was conducted using configurations ranging from 1 to 10 *nodes*, all maintaining a constant 24 *threads* and employing the largest input file, named "1,158,854" (see the first line of Table 1). Fig. 4 illustrates the predicted execution time as a function of the number of *nodes*, alongside the average measured execution time, with each line representing different *bootstrap* values. The model predictions are closely aligned with the measured data. We observed a general decrease in execution time with increased nodes, particularly evident for higher *bootstrap* values. For instance, 5 *nodes* were found to be efficient for *bootstrap* values of 1000 and 2000, whereas additional *nodes* beyond

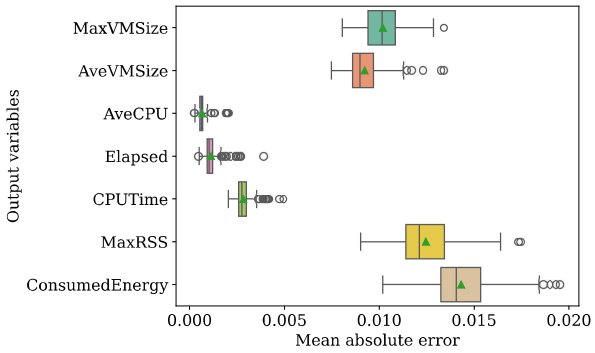


Fig. 3. Boxplot of mean absolute error (MAE) for predicted and database values in regression output variables.

this threshold (e.g., 10 or 20) did not significantly enhance performance. This suggests that utilizing 5 *nodes* optimizes resource utilization and throughput on SDumont for this case study, possibly due to the increased overhead from adding more nodes.

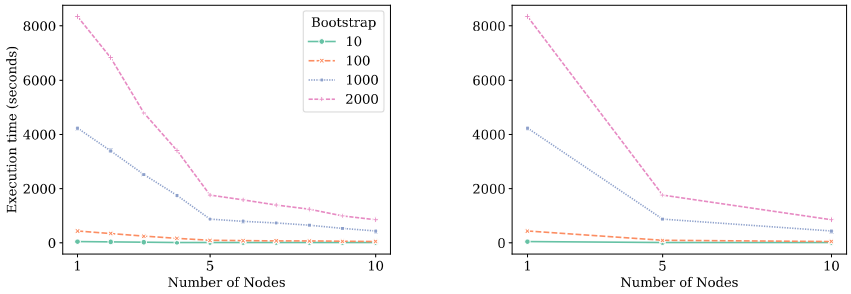
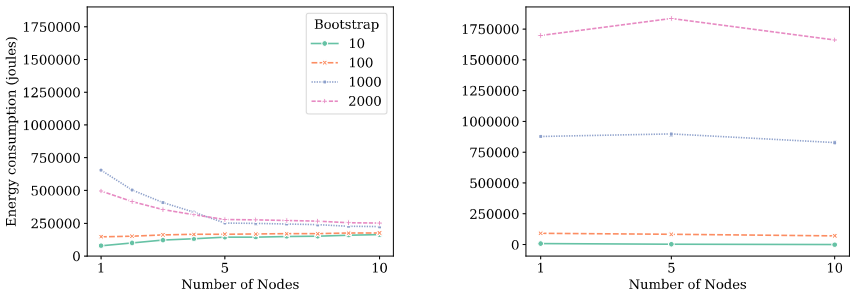


Fig. 4. Application execution time (predicted vs. measured) from 1 to 10 *nodes*.

Fig. 5 shows the predicted and measured energy consumption across different node counts, with each line representing a *bootstrap* value. The model accurately predicts energy consumption for lower *bootstrap* values (10 and 100, which indicate lower computational intensity of the application), closely aligning with measured data. However, predictions were inaccurate for higher *bootstrap* values (1000 and 2000). In contrast to computational time, where execution generally decreases with more nodes, the predicted energy consumption does not follow the same trend. Overall, while the model performs well for lower *bootstrap* values, its accuracy diminishes for higher values.

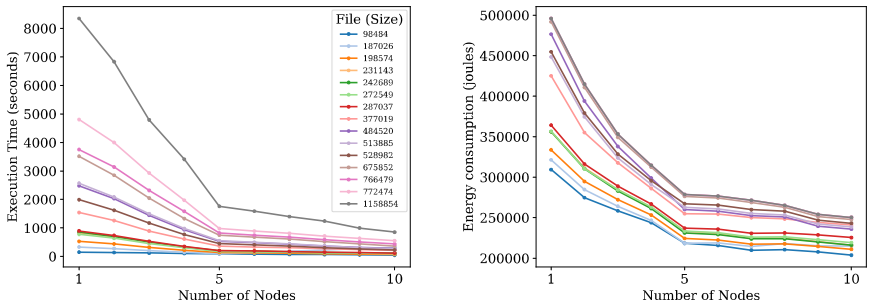
Fig. 6 shows the computational time and energy consumption predictions when varying the input data from real DENV genomes. Fig. 6(a) illustrates that



(a) Predicted (b) Measured

Fig. 5. Application energy consumption (predicted vs. measured) from 1 to 10 nodes.

larger input files benefit from reduced execution times with up to 5 nodes, followed by a gradual decline of up to 10 *nodes*, indicating scalable efficiency. It presents computational time across 1 to 10 nodes using 2000 bootstrap and 24 threads for RAxML. Transitioning from 1 to 5 nodes resulted in an 80% reduction in execution time. Thus, allocating 5 nodes optimizes efficiency and offers significant time savings, which aligns with measured data. Fig. 6(b) demonstrates consistent energy consumption across all dataset sizes. Both larger and smaller datasets benefit from increased nodes up to 5, with a modest decline observed up to 10 nodes, indicating stable energy efficiency across varying dataset sizes.



(a) Predicted Elapsed (b) Predicted ConsumedEnergy

Fig. 6. Application predicted *Elapsed* and *ConsumedEnergy* output variable of the fifteen files with 2000 RAxML bootstrap across 1 to 10 nodes and 24 threads.

Fig. 7 illustrates the relationship between model predictions and measurements computational time (*Elapsed*) and energy consumption (*ConsumedEnergy*), with the number of nodes varying from 1 to 10 and using a *bootstrap* of 2000 for the "1,158,854" input file. The solid blue line predicts a significant reduction in computation time as the number of nodes increases, reflecting an

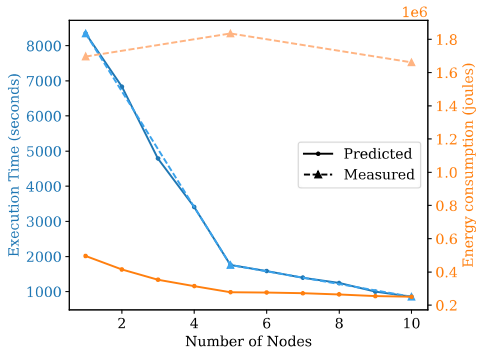


Fig. 7. Comparison between computational time and energy consumption predictions for the input file 1,158,854 with *bootstrap* value of 2000.

improvement in performance with increasing computational resources, which is corroborated by the measurements in the dotted line. On the other hand, the orange line indicates a decrease in energy consumption as more nodes are employed, which could indicate a low computational intensity due to excessive parallelism, but measurements do not validate this. The cause for this inaccuracy is clear when considering Fig. 2, as the ML model currently does not consider the number of nodes as an important parameter to predict *ConsumedEnergy*.

6 Conclusion

As HPC systems become complex, strategies that minimize energy usage while maximizing computational throughput become imperative. A key component to achieve this is ensuring that applications are properly configured to use the available computing resources. This work addresses this issue by providing an ML model that, based on the application's characteristics, predicts execution times and energy consumption for various resources. Thus, this information can be used to allocate the exact resources that maximize performance while limiting excessive energy consumption.

The ML model proposed in this work, based on the Extra Trees Regressor, was trained with extensive datasets from scientific applications like RAxML executions, allowing for informed decision-making in resource allocation and configuration. The model was designed to predict 7 parameters, from which application execution time and energy consumed are the most relevant, based on 4 input variables: bootstrap, which in the context of the case study directly impacts computing intensity; number of nodes used; dataset size; and number of threads used.

Overall, the model is very accurate in predicting the execution time of applications for varying dataset sizes and node configurations, as shown in Fig. 4. The energy consumption prediction is accurate for specific configurations of

the case study, where the computational intensity is lower, as shown in Fig. 5. However, when testing for high *bootstrap* values, the accuracy of the energy prediction is reduced significantly. Fig. 6(b) provides an overview of the predicted and measured values for execution time and energy consumption for the most compute-intensive configuration of the case study.

Moving forward, research and development in ML regression-based prediction aim to address challenges in HPC, such as scalability and real-time adaptation to dynamic workloads. The model, currently operating statically, will be enhanced to better predict energy consumption of applications and adapted for dynamic operation, assessing queue loads in real time to optimize resource allocation. This model will be integrated into an HPC resource scheduler with rigorous validation to ensure efficient and reliable resource allocation and will be validated in a production environment with various scientific software.

7 Acknowledgements

We gratefully acknowledge the generous provision of high-performance computing (HPC) resources by the National Laboratory of Scientific Computing (LNCC) in Brazil and the Santos Dumont supercomputer. This research was partially funded by the INESC TEC International Visiting Researcher Programme 2023 and the National Council for Scientific and Technological Development (CNPq) through the CNPq/MCTI/CT-Biotec project, Grant Number 440360/2022-6.

References

1. Geurts, P., Ernst D. and Wehenkel, L.: Extremely Randomized Trees. Machine Learning **63**(1), p. 3–42 (2006). https://doi.org/10.1007/s10994-006-6226-1
2. Pedregosa, F. , Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and others: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**(85), p. 2825–2830 (2011).
3. Stamatakis, A.: Raxml Vesion 8: A Tool For Phylogenetic Analysis and Post-analysis of Large Phylogenies. Bioinformatics **30**(9), p. 1312–1313 (2014). https://doi.org/10.1093/bioinformatics/btu033
4. Coelho, M., Freire, G., Osthoff, C., Carneiro, A. R., Galheigo, M., Boito, F. Z., Navaux, P. OA, Cardoso, D.: Development of a Machine Learning Framework to Support Efficient Scientific Gateways, Latin America High Performance Computing Conference (CARLA), Porto Alegre (2022).
5. Coelho, M., Freire, G., Ocaña, K., Osthoff, C., Galheigo, M., Carneiro, A., Boito, F., Navaux, P., Cardoso, D.: Desenvolvimento de um Framework de Aprendizado de Máquina no Apoio a Gateways Científicos Verdes, Inteligentes e Eficientes: BioinfoPortal como Caso de Estudo Brasileiro, Anais do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho Florianópolis. Sociedade Brasileira de Computação, p. 205-216 (2022). https://doi.org/10.5753/wscad.2022.226377

6. Yoo, A.B., Jette, M.A. and Grondona, M.: SLURM: Simple Linux Utility for Resource Management, Workshop on Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, p. 44–60 (2003). https://doi.org/10.1007/10968987_3
7. Willmott, C.J., Matsuura, K.: Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance, Climate Research **30**, p. 79—82 (2005). https://doi.org/10.3354/cr030079
8. Ocaña, KA., Galheigo, M., Osthoff, C., Gadelha, Jr LM, Porto, F., Gomes, ATA., Oliveira, D., Vasconcelos, AT.: BioinfoPortal: BioinfoPortal: A scientific gateway for integrating bioinformatics applications on the Brazilian national high-performance computing network, Future Generation Computer Systems **107**, p. 192–214 (2020). https://doi.org/10.1016/j.future.2020.01.030
9. Carastan-Santos, D., da Costa, G., Poquet, M., Stolf, P., Trystram, D.: Lightweight Prediction for Improving Energy Consumption in HPC Platforms. 2024. https://doi.org/hal-04566184
10. James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning. [S.l.], Springer, v. 112, (2013)
11. Matsunaga, A., Fortes J.A.B.: On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications, 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. 2010. https://doi.org/10.1109/CCGRID.2010.98
12. Yokelson, D., Charest M.R.J., Li, Y.W.: HPC Application Performance Prediction with Machine Learning on New Architectures. PERMAVOST '23: Proceedings of the 2023 on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn Strategy, p. 1–8, 2023. https://doi.org/10.1145/3588993.3597262
13. Vercellino, C., Scionti, A., Varavallo, G., Viviani, P., Vitali, G., Terzo, O.:: A Machine Learning Approach for an HPC Use Case: the Jobs Queuing Time Prediction, Future Generation Computer Systems **143**, p. 215–230 (2023). https://doi.org/10.1016/j.future.2023.01.020
14. Balis, B., Lelek, T., Bodera, J., Grabowski, M., Grigoras, C.: Improving Prediction of Computational Job Execution Times with Machine Learning, Concurrency and Computation: Practice and Experience **36**(2), p. e7905 (2024). https://doi.org/10.1002/cpe.7905
15. Sun, J. , Sun, G., Zhan, S., Zhang, J., Chen, Y.: Automated Performance Modeling of HPC Applications Using Machine Learning, IEEE Transactions on Computers **69**(5), p. 749–763 (2020). https://doi.org/10.1109/TC.2020.2964767
16. The MPI Forum: MPI: A Message Passing Interface, Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing, p. 878–883 (1993). https://doi.org/10.1145/169627.169855
17. Yang, Z.: Maximum Likelihood Phylogenetic Estimation from DNA Sequences with Variable Rates Over Sites: Approximate methods. Journal of Molecular Evolution **39**, p. 306–314 (1994). https://doi.org/10.1007/BF00160154
18. SCHEDMD. Slurm Workload Manager: sacct. [S.l.], 2018. Available in: < https://slurm.schedmd.com/archive/slurm-17.02-latest/sacct.html>
19. W. Pfeiffer, A. Stamatakis.: Hybrid MPI/pthreads Parallelization of the RAxML Phylogenetics Code, 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010, pp. 1–8, https://doi.org/10.1109/IPDPSW.2010.5470900
20. Zhou, X., Shen, X.-X., Hittinger, C. T., Rokas, A., Evaluating Fast Maximum Likelihood-Based Phylogenetic Programs Using Empirical Phylogenomic Data Sets, Molecular Biology and Evolution, Volume 35, Issue 2, 2018, Pages 486–503, https://doi.org/10.1093/molbev/msx302