

Who will forecast the forecasting models? Machine Learning approaches to predict meteorological simulations runtime

Allegra De Filippo^a, Emanuele Di Giacomo^b and Andrea Borghesi^a

^aUniversity of Bologna, viale Risorgimento 2, Bologna, 40136, Italy,

^bARPAE, Emilia Romagna, Bologna, 40100, Italy,

ARTICLE INFO

Keywords:

High Performance Computing
Machine Learning
COSMO meteorological model

ABSTRACT

Due to the need of large computing capabilities, predicting the execution time of weather forecast models is a complex task, usually performed on High Performance Computing systems. A reliable prediction allows for an improved planning of the model execution, a better allocation of available resources, and the identification of possible anomalies. In this work, we show how Machine Learning techniques obtain accurate runtime predictions of the COSMO weather forecasting model used at the Hydro-Meteo-Climate Structure of Arpa Emilia-Romagna. Our contribution is twofold: 1) a large public dataset reporting the runtime of COSMO run under a variety of different configurations; 2) a comparative study of ML models, which greatly outperform the current state-of-practice used by the domain experts.

1. Introduction

Estimating weather forecast runtime allows for optimizing the execution planning of complex weather forecasting models such as COSMO. These estimations also allow a better allocation of available resources, as well as to identify anomalies during the execution of the model. A reliable prediction of runtime allows for better management of the overall system, thus providing significant benefits to both system administrators and users. In this context, we show how Machine Learning (ML) techniques are able to obtain accurate results for runtime predictions of the COSMO meteorological model for Small-scale Modeling (a) (Consortium for Small-scale Modeling) in the configurations managed by the Arpa Hydro-Meteo-Climate Structure (SIMC) and performed on GALILEO, High Processing Computing (HPC) system hosted at CINECA CINECA (2015).

Numerical weather forecast models simulate the evolution of atmospheric weather and they are typically used for the production of weather forecasts, one or even more times a day. These models often require large computing capabilities, such as HPC systems Fuhrer, Osuna, Lapillonne, Gysi, Cumming, Bianco, Arteaga and Schulthess (2014), and it may become difficult to identify execution problems. For this reason, reliable estimates of the time to solution for a given set of simulations are very hard to obtain. Due to their complexity, typically the COSMO models are run in a fixed configuration and with defined runtimes. However, an accurate forecast of the runtime allows a better allocation of the available resources, an easier identification of the optimal configuration, and a rapid and effective reallocation of the available resources (e.g., by following a decrease in the number of nodes due to failures, with the introduction of new tasks, or with the re-execution of previous instances following malfunctions). Furthermore, in case of excessive

deviations from the observed time, the predictor can identify anomalies that arise during the execution of the model.

Recent works showed how learning models Pittino, Bonfà, Bartolini, Affinito, Benini and Cavazzoni (2019) allow obtaining accurate results for predicting the runtimes of scientific applications run on HPC systems. Based on this idea, here we show that accurate predictions can be obtained without analyzing the application code itself but based only on a selection of configuration parameters and observed times, and we focused on the complex application area of numerical meteorological modeling.


The first issue to tackle in this application domain has been the absence of available historical data. Due to this, it has been necessary to create a dataset by identifying the most representative parameters, running the model according to different configurations, and collecting the results. This data collection represents an essential initial step for this work and a useful resource for the model area of Arpa-SIMC in analyzing the model performance. For the definition of the relevant parameters, the support of a domain expert was used.

Once obtained this dataset, we conducted a preliminary comparative analysis of different regression models for predicting the execution time of COSMO models, by analyzing the time/quality trade-off of these regression models. Finally, we show how ML solutions allow for greater accuracy than the predictions obtained with a parametric formula based on the operational version of COSMO.

The main contributions of the paper are the following:

- a large public data set reporting the runtime of the COSMO meteorological model run under a variety of different scientific parameters and parallelization levels. The dataset is public and available here: <https://zenodo.org/record/5818362>;
- a comparative study of Machine Learning models to accurately estimate the runtime of the weather forecast model, which overall greatly outperforms the current

*Corresponding author

 allegra.defilippo@unibo.it (A. Borghesi)
ORCID(s):

state-of-practice used by the domain experts to predict the simulation duration.

The rest of the paper is organized as follows. Section 2 provides a brief description of HPC systems and meteorological models, in particular, COSMO and its use at Arpa-SIMC, highlighting the parts of interest for the purpose of creating tools useful for forecasting the runtime. Section 3 is dedicated to the dataset necessary for the training of Machine Learning models: the definition of parameters and their values, the selection of the configurations to be run, and finally the execution of the models to be able to collect data. Section 4 describes the models taken into consideration for the implementation of predictors of COSMO-Model runtimes. Section 5 describes all the experiments carried out and the comparisons between the various models involved. Final remarks are in Section 6.

2. Related Work

Numerical weather forecast models are mathematical models that allow you to predict the trend of atmospheric conditions in a certain area, simulating the trend through a schematic representation of physical reality based on a set of differential equations Holton (1973). Since it is not possible to obtain exact solutions of the differential equations, a numerical approximation is used which allows obtaining discrete solutions, for example by means of the finite difference method. This discretization allows the representation of space and time by using cells of a three-dimensional grid (latitude, longitude, altitude) and equidistant instants: the simplification consists in considering, therefore, uniform atmospheric conditions within each cell that varies only in the passage from a certain instant to another. Accordingly, the resolution of the model depends on the size of the cells (the so-called grid step), which is inversely proportional to the spatial approximation; and higher resolution means more computational effort.

Given their computational complexity, these models require large computing capabilities, such as High-Performance Computing systems. Generally, in HPC systems, each application has its own intrinsic limitations in terms of scalability: beyond this threshold, runtimes do not improve or even may deteriorate, as the increase in complexity in communication between nodes causes a degradation in performance. Similarly, also the number of cores per node can affect the execution time. For these reasons, predicting the runtime of applications such as COSMO models is a complex but potentially extremely useful task.

In literature, many recent works focused on estimating the application workflows using different machine learning techniques, respectively, decision tree methods Miu and Missier (2012), neural network methods Nadeem, Alghazawi, Mashat, Fakeeh, Almalaise and Hagrass (2017). An extensive survey on prediction models of applications for resource provisioning is presented in Amiri and Mohammad-Khanli (2017). All these approaches mainly focused on monitoring the online status of the machine, instead of exploiting

the domain knowledge of the application and its input. Machine Learning models have been proposed to estimate the duration of different classes of algorithms, often for the goal of selecting and configuring algorithms for a given task. For this purpose, models describing the relationship between the configuration of an algorithm and its performance have proven to be very useful Hutter, Hamadi, Hoos and Leyton-Brown (2006). For instance, Hutter et al. Hutter, Xu, Hoos and Leyton-Brown (2014) evaluate a series of ML methods for predicting algorithms runtime under different configurations.

Broadly speaking, the methods described so far cover only algorithms run on single consumer-grade computing nodes and with off-the-shelf hardware; moreover, the considered algorithms tend not to be computationally intensive. In the high-performance and scientific computing context, estimating workload runtime is still a less explored area. The work of Pittino et al. Pittino et al. (2019) in the field of materials sciences showed how Deep Learning models allow for obtaining very accurate results for predicting the runtimes of scientific applications on HPC systems. They show that accurate predictions can be obtained without analyzing the application code itself but based only on a selection of configuration parameters and observed times.

In this work, we adopted a similar strategy and we show how ML techniques are able to obtain accurate results for runtime predictions in a complex application domain, such as the COSMO forecasting model used at the Hydro-Meteorological Structure of Arpa Emilia-Romagna. Due to the absence of historical data, it has been necessary to create a dataset by identifying the most representative parameters, running the model according to different configurations, and collecting the results. This data collection represents an initial step for this work, but also a useful resource for the COSMO model area in analyzing its performance.

3. The Cosmo Model and the Dataset

In this section, we start by describing the COSMO weather forecast simulation model, and we then provide exhaustive details on the dataset creation process.

3.1. COSMO Model

The limited-area forecast model considered in this paper is called *COSMO* (CONsortium for Small-scale MODELing) for Small-scale Modeling (a); Schulthess, Bauer, Wedi, Fuhrer, Hoefler and Schär (2018) and it is widely used both for weather forecasts and numerous scientific applications¹. The COSMO model is a non-hydrostatic limited-area atmospheric prediction model Steppeler, Doms, Schättler, Bitzer, Gassmann, Damrath and Gregoric (2003). It is based on the primitive thermo-hydrodynamical equations describing compressible flow in a moist atmosphere. The model equations are formulated in rotated geographical coordinates

¹See the consortium publication list for an exhaustive overview at <http://www.cosmo-model.org/content/model/documentation/journals/default.htm>

and a generalized terrain following height coordinates. A variety of physical processes are taken into account by parameterization schemes for Small-scale Modeling (b). As exact solutions are computationally intractable, the COSMO model relies on numerical approximations to reach discrete solutions. This discretization involves the representation of space and time under the form, respectively, of cells of a three-dimensional grid (latitude, longitude, height) and equidistant time-steps: the simplification consists in considering uniform atmospheric conditions within each cell that varies only in the passage from one time-step to another. It can be deduced from the size of the cells (the so-called grid step) depends on the resolution of the model, which is inversely proportional to the spatial approximation.

Given its computational complexity, the COSMO model is generally run in parallel mode: each process involved solves the equations of the model within a regular subdomain, obtained from the decomposition of the considered spatial domain. The various processes associated with contiguous subdomains communicate by exchanging messages as they need the values in the cells positioned just beyond their borders. Consequently, a high number of processes leads to a greater parallelization but also a greater number of messages exchanged. The subdivision is made only along the latitude and longitude: given a spatial domain of $n_i \times n_j$ cells (respectively along the longitude and latitude) and chosen the number of processes p , the possible decompositions are even to the pairs of divisors of p : in fact, to obtain p subdomains we must divide the grid along the x-axis in n_x parts and along the y axis in n_y parts, with $n_i \times n_j = p$. The resulting subdomain size is $d_x = \frac{n_i}{n_x}$ and $d_y = \frac{n_j}{n_y}$.

Moreover, if the model is run on a cluster architecture, it is also necessary to decide how to divide these processes among the nodes and how many cores per node to use: some studies on the performance of the COSMO model² have shown that with the same processes but with a different number of nodes (and consequently cores per node), the use of more nodes (and consequently fewer cores per node) leads to an improvement in performance. One of the biggest bottlenecks is memory bandwidth. Furthermore, the model can be executed in single or double precision, thus using floating point numbers of 4 and 8 bytes respectively, and the use of single precision involves a significant increase in performance (i.e., decreased runtime), in exchange for accuracy reduction.

3.2. Dataset Definition and Creation

A substantial difference among COSMO configurations concerns the size and temporal step of the grid. To these two parameters, we can add the number of nodes, the number of cores per node used, and the number of processes.

- Fixed, 10.0 × 10.0 degrees Grid Dataset (DGF)
- Variable Grid Dataset (DGV)

The two datasets have the same parameters, except for the size of the grid in the case of DGF: this allows to obtain a dense dataset, in which the space is explored in a more exhaustive way.

Tables 1 and 2 show the parameters and ranges used, on which some assumptions can be made. The grid_step value is chosen among the values compatible with the fixed grid. The same values were used for DGV in order to have two homogeneous datasets and to be able to reuse the DGF data in DGV. The ranges n_nodes and n_cores have been identified among the configurations normally used and by respecting the number of computing resources available for Arpa-SIMC. Furthermore, the number of cores is always even, and the cores used are equally divided. The relationship between x and y of the subdomain_ratio has been limited only to the interval that, realistically, is used: the stretchier shape of x leads to better performance, but a too-unbalanced relationship leads to an increase in the exchange of messages between processes. The lower bounds of x_length and y_length were chosen based on a minimum value to obtain a meaningful prediction. Similarly, the upper bounds are related to the availability of the initial data that allow starting the execution of the model, since Arpa-SIMC considers a specific area (that of the COSMO-5M operating model, see Figure 1). Each experiment with a variable grid is therefore limited in this domain, which is an area that covers approximately the whole Mediterranean.

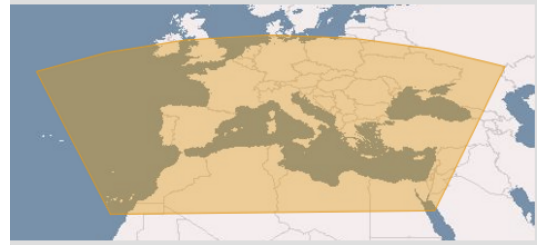


Figure 1: COSMO-5M domain

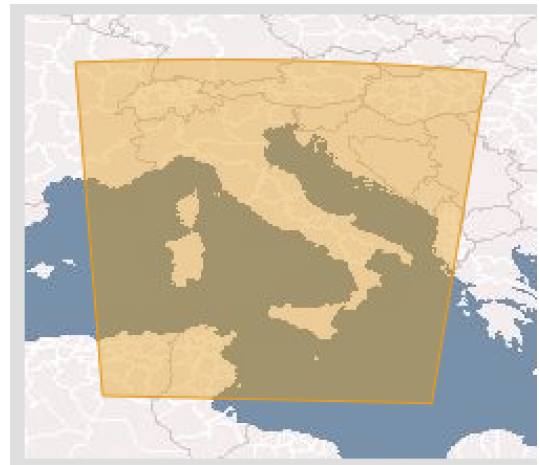


Figure 2: COSMO-2I domains

²<http://www.cosmo-model.org/content/tasks/pastProjects/pompa/default.htm>

Parameter name	Parameter description	Values
grid_step	Cell size (degrees)	0.02, 0.05, 0.08, 0.1
n_nodes	Number of nodes	[10, 60]
n_cores	Number of cores per node	[14, 32] (even)
single_precision	Use of single precision	boolean
subdomain_ratio	Ratio between x and y	[1.0, 5.0]

Table 1

Parameters of DGF dataset

Parameter name	Parameter description	Values
x_length	Cell size x (degrees)	[7.0, 48.0]
y_length	Cell size y (degrees)	[7.0, 25.0]
grid_step	Cell size (degrees)	0.02, 0.05, 0.08, 0.1
n_nodes	Number of nodes	[10, 60]
n_cores	Number of cores per node	[14, 32] (even)
single_precision	Use of single precision	boolean
subdomain_ratio	Ratio between x and y from subdomain	[1.0, 5.0]

Table 2

Parameters of DGV dataset

We can also observe that the parameters do not include the number of forecasting hours: the processing at each hour is based on the data of the previous instant and is therefore sequential. The forecasting process of each hour always requires the same time, with the exception of the first hour which has a significantly longer duration: in the case of the COSMO-2I operational configuration, it is about 20% more. To limit the resources dedicated to the series of experiments described in the following, we run each configuration of the model only for the first two hours of forecasting, respectively $time_1$ and $time_2$, as the simulation of the hours following the second one requires always a time equal to $time_2$; $time_1$ and $time_2$ are stored after each run and they constitute the values which will be predicted with the ML models. The total running time of a model per N forecast hours can then be calculated with the following formula:

$$time_{tot} = time_1 + (N - 1) \times time_2 \quad (1)$$

3.3. Selection of Model Configurations

Starting from the selected values in Tables 1 and 2, it is possible to enumerate all the model configurations that can be generated for both datasets: the number of grid steps is finite and, for each pair of <number of nodes, number of cores per node>, there is a number of possible subdomains. In order to plan the experiments, a threshold of 500 seconds per forecast hour has been set: it represents a right upper bound for the fully operational time required (e.g., COSMO-5M would be completed in about 10 hours), but it is necessary to include all the day forecasts past configurations, which do not require tight deadlines. To estimate the runtime, COSMO experts typically used Equation (2), based on the COSMO-2I model. In the rest of the paper, we will refer to Equation (2) as a Pessimistic Prediction (PP).

$$time^{PP} = time_F \times \frac{step_C}{step} \times \frac{area}{area_C} \times \frac{nproc_C}{nproc} \times (1 - 0.3 \times sp) \quad (2)$$

In the equation above, $time_F$ represents the average runtime of the first forecast hour of COSMO-2I, $step_C$ and $step$ are the grid steps of COSMO-2I and of the configuration in question respectively (grid_step), $area_C$ and $area$ are the areas respectively of COSMO-2I and of the configuration under consideration (product of x_length and y_length), $nproc_C$ and $nproc$ are the number of processes respectively of COSMO-2I and of the configuration under examination (product of n_nodes and n_cores), sp is 1 if the configuration under examination is in single precision, otherwise 0: experimentally, it has been found that single precision reduces runtime by about 30%. Corresponds to the single_precision parameter. The average runtime of the first hour of the COSMO simulation ($time_F$) represents a form of domain knowledge possessed by the expert; it can also be measured using the dataset provided as a contribution of this work. In the case of DGF, the number of possible configurations is equal to 5758 and, since they all have an estimated time of fewer than 500 seconds, none have been discarded. In the case of DGV, however, only 892 out of 42384 (2%) are beyond the threshold. The two additional parameters of DGV (x_length and y_length) require sampling: therefore 5 different domains have been identified (one of which is that of DGF, i.e. 10.0 x 10.0 degrees) whose dimensions have been chosen trying to maintain a uniform distribution in terms of pairs of values, the relationship between the two and area. This distribution was chosen to try not to over-represent any configuration (see Figure 3).

For each of the five domains, 1000 configurations were randomly selected, for a total of 5000 elements. Random

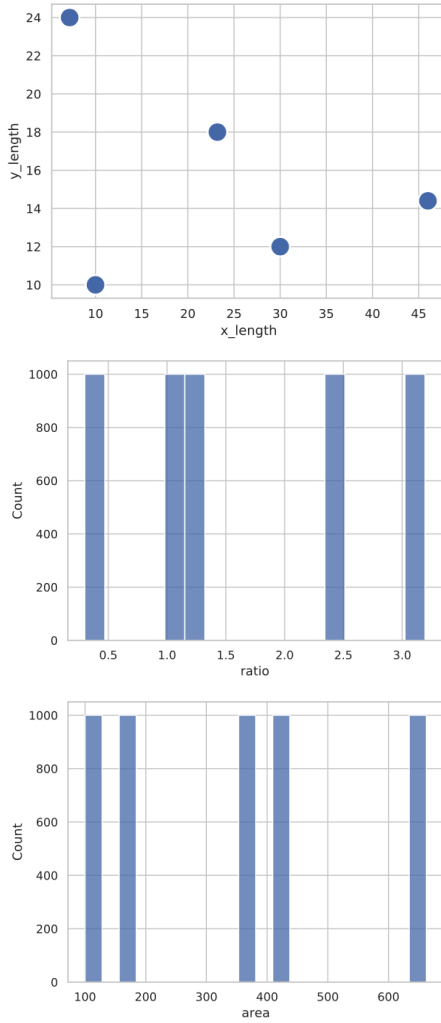


Figure 3: (up) Size of domains selected for DGV, (center) ratio between x_length and y_length of the domains selected for DGV, (down) area of domains selected for DGV

sampling was chosen to respect the distribution of the various parameters within each domain.

3.4. Model Configurations and Data Collection

The configurations were performed on GALILEO³, in the same environment as the operational versions of Arpaes-SIMC and using the same tools. In particular, a series of scripts have been implemented in Python to generate each instance of the model starting from the configuration of the COSMO-2I operating model and modifying only the fields relating to the chosen input parameters. The same tool was also used for scheduling the various instances of the model, namely Simple Linux Utility Resource Management (SLURM) Yoo, Jette and Grondona (2003), a popular open-source system for managing cluster resources. This tool was useful not only to exactly replicate the original environment, but also to avoid overlapping the daily operating hours of the operating model and to run the experiments only in the

time windows of inactivity: this point was addressed with particular attention, in order to avoid cascading effects on the entire forecasting chain.

Each configuration was performed in exclusive mode (SLURM –exclusive option), without sharing the node with other jobs. On the contrary, the runtimes would have been distorted because, as already stated, the number of cores per node affects the performance of the model. At the end of each execution, we store the times required by each phase of the model execution, including those required for the generation of each forecast hour ($time_1$ and $time_2$).

Since the error on $time_2$ has a greater impact on the accuracy of the overall forecast (see Equation (1)) and because PP generally tends to overestimate (see Figure 4), a second estimate is often considered by domain experts as well, identical to PP except for the replacement of average $time_F$ (Equation (2)) of the first simulation hour of COSMO-2I with the second ($time_S$), about 20% lower. It is called Optimistic Forecasting (OP):

$$time^{OP} = time_S \times \frac{step_C}{step} \times \frac{area}{area_C} \times \frac{nproc_C}{nproc} \times (1 - 0.3 \times sp) \quad (3)$$

By comparing the cumulative functions of the absolute and relative error in the prediction of $time_2$ by PP and OP on DGV (Figure 5), we can see how OP is more accurate and in particular how all predictions have a relative error less than 100%, compared to 140% of PP. The same considerations apply to DGF, where the behavior of PP and OP is similar.

Concerning DGV, all the times observed (both $time_1$ and even more so $time_2$) are below the threshold of 500 seconds, respectively 418.45 and 383.19 seconds.

Furthermore, analyzing the DGV dataset, it is evident that the ratio between the size of the subdomain (subdomain_ratio parameter) has a limited influence on the runtime. The coefficient of variation of $time_2$ for each group of configurations that differ only in subdomain_ratio has an average value of 2.3% and is at most 10% in 98% of cases.

4. Methods

The tackled problem is a regression one: given the features describing the execution of the COSMO model we want to predict the model runtime. As previously seen, this is not a straightforward matter, as the total runtime is the composition of the $time_1$ (the time for the first hour of simulation) and $time_2$ (the time for the second hour of the simulation – equal to all subsequent hours). As *targets* for the regression tasks we consider all the involved times, namely $time_1$, $time_2$, and the total time⁴. This means that we will create a different regression model for each of these three different targets. Moreover, after a preliminary evaluation and a discussion with domain experts, we realize that the behavior of the COSMO model with the fixed grid is significantly different compared to the variables grid; hence,

³<https://www.hpc.cineca.it/hardware/galileo100>

⁴See Eq. 1

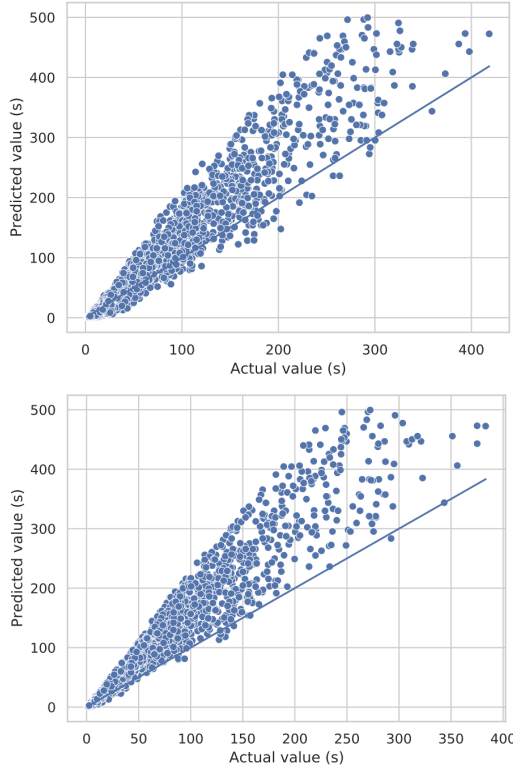


Figure 4: Correlation between observed and estimated time using PP for DGV, $time_1$ (left) and $time_2$ (right).

we create two different sets of ML models, respectively dealing with the fixed and the variable grids. In the end, we considered the following types of regression problems: 1) $time_1$ for simulation with fixed grid, 2) $time_2$ with fixed grid, 3) total time with fixed grid, 4) $time_1$ for simulation with variable grid, 5) $time_2$ with variable grid, 6) total time with the variable grid.

For each of these six classes of regression task, we chose a selection of ML and DL models, in addition to the optimistic estimate (OP) described in Section 3.3 (see Eq. 3 in particular). The optimistic estimate is the heuristic currently used by the researchers using the COSMO model to schedule their activity and to request HPC resources to run their simulations, thus we consider it as our baseline. As already shown, domain experts use as well a pessimistic estimate (Eq. 2) but it produces a less accurate estimate, thus we decided to only focus on the best domain-based forecast. The ML/DL models selected are the following:

- Linear Regression (LR) Chatterjee and Hadi (2013)
- Decision Tree regressor (DT) Breiman, Friedman, Stone and Olshen (1984)
- Random Forest (RF) Breiman (2001)
- Gradient Boosting (GB) Friedman (2001)
- Fully Connected Neural Network (FCNN) Bengio, Goodfellow and Courville (2017).

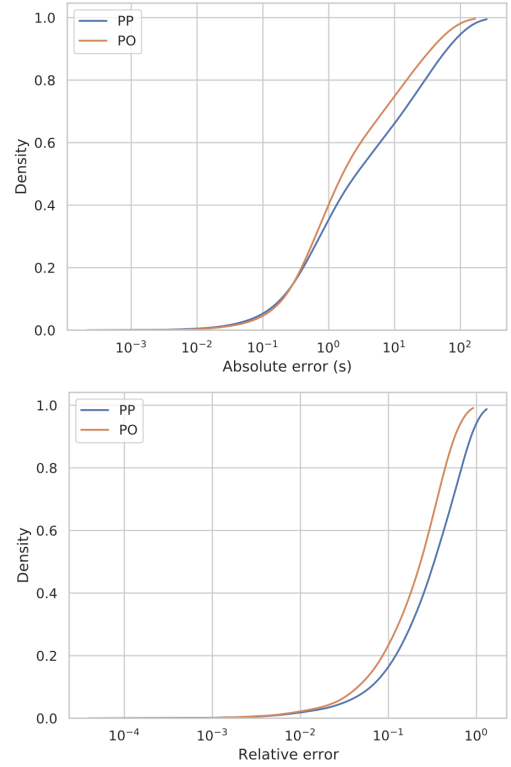


Figure 5: Pessimistic prediction for $time_1$ (up) and $time_2$ (down)

For the standard ML models (LR, DT, RF, and GB) we performed a preliminary empirical evaluation to identify the optimal parameters; for instance, in the RF case, the best results are obtained with a random forest composed of 80 estimators, while the decision tree optimal depth is equal to 20. In the case of the FCNN, given the vastly larger architectures space, we performed hyperparameters fine-tuning via Bayesian optimization, using the Mean Average Error (MAE) computed on a validation data set as the objective function to be minimized. The domain space to be explored was identified after a preliminary empirical analysis (e.g., topological hyperparameters, such as the upper and lower bounds on the number of layers, and optimization-specific hyperparameters such as the learning rate and batch size).

5. Experimental Results

In this section, we experimentally evaluate the ML models for the prediction of COSMO model runtime. The ML models were implemented using the python library scikit-learn Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot and Duchesnay (2011), the DL model using the pytorch Paszke, Gross and et al. (2019), and for the hyperparameter tuning we used hyperopt Bergstra, Yamins and Cox (2013).

We will start with the analysis of the results for the specific components of the weather forecast simulation, $time_1$, and $time_2$. Afterward, we focus on the prediction of the total time required to execute COSMO. In the following, we will

Grid	Model	$time_1$			$time_2$		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE
Fix.	OP	7.2	11.9	33.3	7.3	13.3	28.3
Fix.	LR	4.9	8.9	27.7	3.4	6.7	25.7
Fix.	DT	2.4	4.5	15.3	0.8	2.1	3.9
Fix.	RF	1.8	3.5	12.3	0.6	1.8	3.1
Fix.	GB	2.3	4.3	15.0	0.7	2.1	3.8
Fix.	FCNN	1.6	3.4	8.3	0.6	1.8	2.8
Var.	OP	24.4	48.4	67.9	9.8	21.6	27.1
Var.	LR	12.5	29.0	36.7	10.2	25.3	38.8
Var.	DT	3.7	7.3	16.2	2.3	6.1	7.1
Var.	RF	2.7	5.2	12.4	1.6	4.3	5.2
Var.	GB	3.7	7.1	16.2	2.2	5.8	6.7
Var.	FCNN	2.3	4.3	10.3	1.0	2.9	3.8

Table 3

Prediction errors with targets $time_1$ and $time_2$; fixed ("Fix.") and variable ("Var.") grid.

consider the ML models described previously (Sec. 4) using their acronyms. In all experiments we performed 10-fold cross-validation; we report only the average values over all folds, as the difference between the different folds was negligible. For all experiments we also normalize the data in the $[0,1]$ range; while this is not strictly necessary for methods such as DT, RF, and GB, we perform this normalization to ensure a fair comparison with the FCNN (neural networks do need normalized data).

5.1. Specific Phase Runtime Prediction

We start by considering how the ML models fare with the prediction of the individual phases involved in the execution of the COSMO model, that is $time_1$ and $time_2$, described in Sec. 3.2. As a reminder, $time_1$ and $time_2$ represent, respectively, the time needed to run the first hour of the forecast model simulation and the time required to simulate the second hour (all subsequent simulated hours require the same time as $time_2$). In Table 3 we report the results, presenting those obtained with both the fixed and variable grid (identified by the first column on the left). The second column specifies the ML model (see Sec. 4); as the OP. The following columns are divided into two groups, the first three being the evaluation metrics computed for the prediction of $time_1$ and the second three pertaining to $time_2$. As evaluation metrics, we employ the Mean Absolute Error (MAE), the Root Mean Square Error (RMSE), and the Mean Absolute Percentage Error (MAPE)⁵, which is expressed as a percentage. For all the metrics, lower values are to be preferred, as lower errors indicate higher prediction accuracy.

The results are quite neat: with the exception of the linear regression, all ML models undoubtedly predict the duration of the first ($time_1$) and second ($time_2$) simulation hours much more accurately than the optimistic prediction (OP), which is already the best estimate currently adopted by domain

⁵The Mean Absolute Percentage Error is computed as $MAPE = 100 \frac{|y_{true} - y_{pred}|}{y_{true}}$, where t_{true} and t_{pred} represent the ground truth and the prediction.

experts. More in particular, with the fixed grid, FCNN slightly outperforms all other ML models but the random forest is very close; instead, in the case of the variable grid (a more complicated learning task), the FCNN takes a more decisive lead, which marked improvements in terms of all considered metrics. The failure of the linear regression is most probably due to the fact that the underlying function to be approximated (the relationship between $time_1$ and $time_2$ with the features describing the COSMO model simulation) is not linear at all.

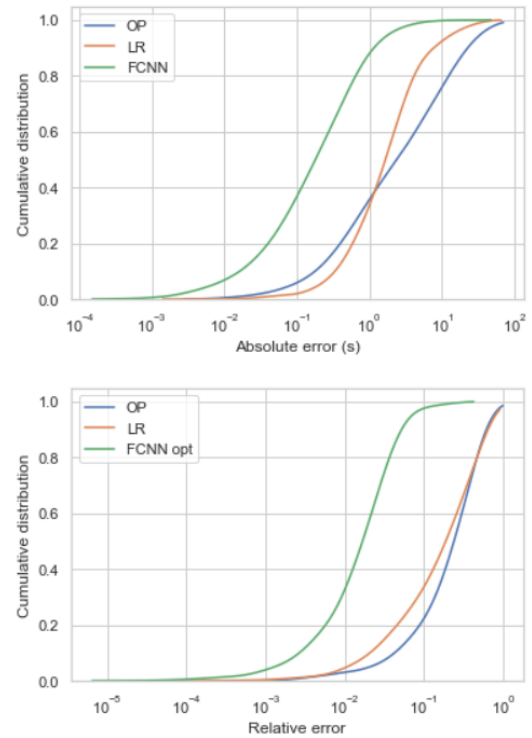


Figure 6 corroborates the experimental evaluation quantitatively described by Table 3 by plotting the cumulative error distributions for $time_2$, that is the most relevant target as it represents the duration of *all* simulation hours after the first

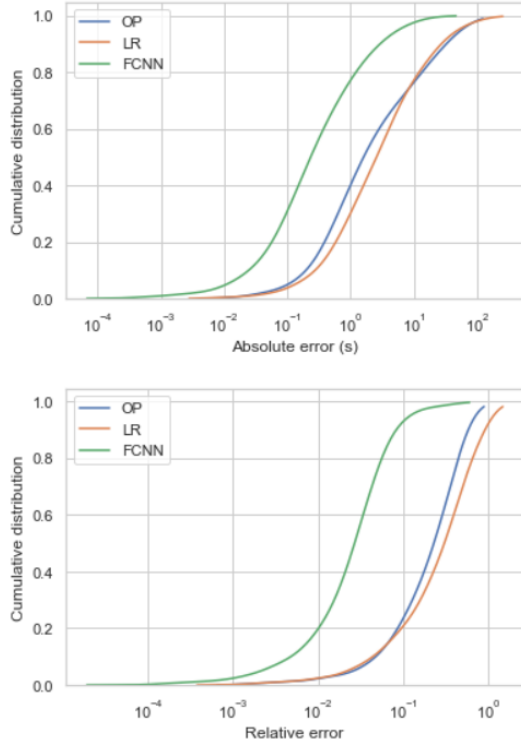


Figure 6: Cumulative error distribution for $time_2$, selected prediction models. Fixed grid (left) and variable grid (right).

one (hence it has a larger impact on the overall execution of the COSMO model). We decided to highlight only a subset of the considered ML models, not to clutter the figures; in particular, the error distribution for the optimistic estimate, the linear regression, and the FCNN are presented. It is very easy to see how for both fixed and variable grid simulations the FCNN significantly outperforms the linear model and the domain experts' estimate (OP). This further indicates that the potential benefits of adopting ML approaches are substantial.

5.2. Total Runtime Prediction

Now we focus on the prediction of the total time required by the COSMO model, again considering both fixed and variable grid simulations. The results are presented in Table 4, considering the same prediction approaches as before and again considering fixed and variable grids separately. In this case, we present only one set of accuracy metrics, related to the total time.

There are two main observations that can be made. First, the ML models vastly outperform the domain expert estimate (OP) in this case as well; as expected from the results discussed previously, the linear regression model is not suited to this learning task, with an accuracy on par with the optimistic estimate. Secondly, there is a significant difference regarding the top-performing ML model. While with $time_1$ and $time_2$ the FCNN was the most accurate model, for the total runtime of COSMO the best method is clearly the random forest, closely followed by the decision tree. The

Grid	Model	MAE	RMSE	MAPE
Fix.	OP	479.3	1037.0	27.7
Fix.	LR	383.7	679.8	48.4
Fix.	DT	20.8	42.8	2.4
Fix.	RF	17.0	36.5	1.9
Var.	GB	66.2	159.4	5.3
Fix.	FCNN	33.7	126.0	2.9
Fix.	OP	1410.5	3536.4	58.8
Fix.	LR	873.5	2040.3	56.5
Var.	DT	33.7	76.2	2.4
Var.	RF	22.6	55.9	1.7
Var.	GB	205.2	624.1	8.6
Var.	FCNN	52.6	162.7	3.4

Table 4

Prediction errors with the different ML models for the total runtime as target; fixed ("Fix.") and variable ("Var.") grid.

neural network has a middle-of-the-pack performance, highlighting that it struggles to obtain the accurate predictions which characterized the specific phases' duration.

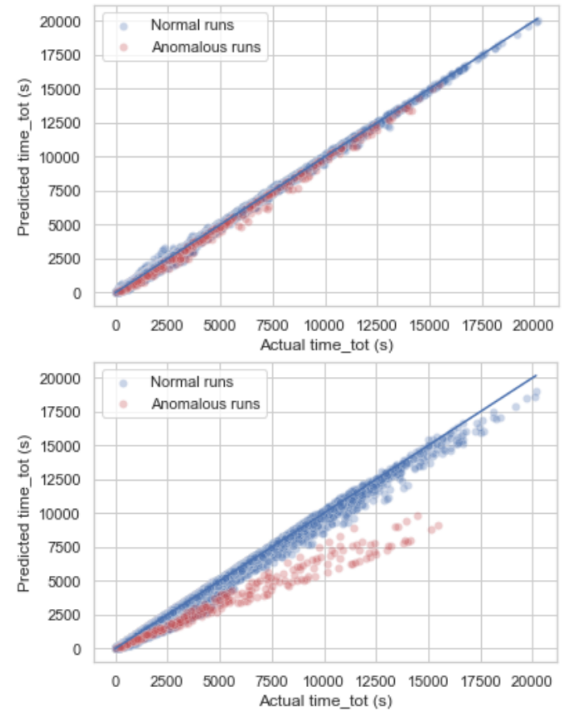


Figure 7: Actual total runtime versus predicted time, RF (left) and FCNN (right).

We decided to better investigate these unexpected results, as the neural network proved to be the most accurate method for the prediction of the single phases. Thus, we looked at the error distributions, focusing in particular on the FCNN and the best model for predicting the total time, namely the random forest. After a thorough manual inspection, we realized that there exist a group of jobs (the vast minority) that possess longer duration than their

counterparts with similar scientific and parallelization parameters. We identified no exact cause of these delays by analyzing the log files, due to the multiple possible explanations. However, their identification could be greatly useful to prevent anomalous situations. For example, the cause could be HW resource contention, where multiple jobs might share the same computing node Hood, Jin, Mehrotra, Chang, Djomehri, Gavali, Jespersen, Taylor and Biswas (2010) if it is not fully occupied, or communication issues between the parallel processes due to networking problems. In summary, we can define these jobs (these runs of the COSMO-model) as being *anomalous*, at least in this specific context. The majority of jobs (> 95%) exhibit normal behavior, thus the anomalous ones can be considered outliers in the training data.

These anomalous jobs seem to be harder to handle for the FCNN, as demonstrated by looking at the prediction errors (on the test set) divided by job type (normal versus anomalous). Fig. 7 shows the predicted total runtime versus the actual one with the RF and the FCNN, using a different coloration for the jobs with a “normal” duration (blue dots) and the slower ones (red dots); we chose to show only the RF as it is the top performing method for the total time prediction. Perfectly predicted jobs follow the diagonal line, while jobs below (above) the line correspond to underestimates (over-estimates). It is easy to notice how the RF clearly has a better performance compared to the FCNN, and this is especially true for unexpectedly long jobs, while the neural network struggles to predict them correctly, always underestimating their actual runtime. This visually striking different behavior corroborates the aggregate (mean) differences reported in Table 4.

Considering the data at our disposal, the reason for the behavior of the FCNN is not entirely clear. A possible explanation lies in the hyperparameter fine-tuning which has been performed on the neural network, an operation not done for the standard ML models. The fine-tuning via Bayesian optimization could have forced the FCNN to overfit on the majority class seen in both training and validation sets (the jobs with normal duration), thus hindering its capability to actually estimate the duration of the overlong jobs, as shown in the predictions over the test set. Finally, it must be noted that while, on average, the RF and DT models are to be preferred to estimate the total duration of COSMO-model simulation, the FCNN’s inability to obtain low prediction errors with anomalous jobs might actually come in handy when doing a *post-mortem* analysis, as the FCNN can identify (classify) jobs with slower-than-expected duration by looking at the log files, after the job completion. This might be useful in terms of HPC resources accounting and pricing computation, as these aspects are typically strictly tied to the job duration Borghesi, Bartolini, Milano and Benini (2019); moreover, this kind of analysis could be regularly performed by system administrators to identify potential issues or bottlenecks causing unexplained workload slowdowns Bartolini, Beneventi, Borghesi, Cesarini, Libri,

Benini and Cavazzoni (2019). On the other hand, the management of HPC resources can be improved by taking proactive scheduling and resource allocation decisions (e.g., maximizing job throughput and guaranteeing low wait times) at dispatch time if accurate duration estimates are available for all submitted jobs Galleguillos, Sîrbu, Kiziltan, Babaoglu, Borghesi and Bridi (2017); in this case, the best ML models are those with the highest average accuracy, such as the RF.

5.3. Discussion

As clearly highlighted by the experimental evaluation, using ML models we can definitely obtain very accurate estimates of the runtime of the COSMO model, in fact, the predictions are significantly more accurate than those currently used by domain experts and practitioners using their knowledge. There is no clear winner in terms of performance among the various ML models, albeit a clear difference emerged: the old-school ML models such as Gradient Boosting and Random Forest tend to perform better in the case of fixed grid, while with a variable grid the Deep Neural Networks provide the most accurate results. This might be related to the more complex behavior of the variable-grid simulations, which involve as well a large number of input features, rendering the relationship to be learned more complex; thus DL models with their capability of approximating any function have a neat advantage.

In complex systems like COSMO, job scheduling and resource allocation are carefully managed to optimize the usage of expensive hardware resources. Based on the available HW, usually fixed hardware configurations and fixed runtimes are used, but we show in our results that we can tackle this issue by accurately predicting the runtime of these models, with a consequent better allocation of the available resources. Indeed, we can observe a double advantage in predicting runtime for these systems: 1) the possibility to improve scheduling, job computational pricing, and the energy efficiency of the whole system; 2) a better matching of HW resource allocation, based on computational needs. Thus providing great benefits to both system administrators and final users. In particular, based on our results, we can conclude that the provided analysis of different ML models could be useful for both a facility owner and a final user: in the first case, better accuracy in runtime predictions (the case of RFs results) could help to improve job scheduling and resource allocation of the entire system; while for a final user, a posteriori analysis could help to identify anomalous runs (this is the case of FCNN results). Moreover, this benchmark represents a further contribution to the field, since it could be integrated with optimization techniques to provide optimal HW resources, based on user-defined constraints.

6. Conclusions

In this paper, we tackle the issue of predicting the time required to execute a complex model for weather forecasts running on HPC resources. In particular, we consider the COSMO model, the meteorological simulation model used

to prepare weather forecasts in the Emilia-Romagna region, Italy. As this model i) needs to be run many times every week (once every night for short-term predictions and at least twice a week for medium- and long-term forecasts) and ii) it occupies a non-negligible amount of computing resources (from a few dozen to hundreds of nodes), it represents a very significant workload for the supercomputer used for the simulation (the tier-0 Marconi100 system hosted at CINECA, the Italian supercomputing center). From this significance stems as well the importance of carefully scheduling and managing the COSMO model. The possibility to forecast its runtime is fundamental for optimized scheduling and better pricing schemes.

In this work, we treat COSMO model simulation as a “black-box”, without exploiting any information about its internal working and phases. Thus we opted for predictive models (from the ML area) without specific adaptations, e.g., standard fully-connected neural networks. In future works, we will explore the possibility to exploit domain information (e.g., Borghesi, Baldo, Lombardi and Milano (2020); De Filippo, Borghesi, Boscarino and Milano (2022); De Filippo, Lombardi and Milano (2019); De Filippo and Borghesi (2022)), either via enriching the feature space or sampling different and more informative configurations to obtain the dataset or using a specialized architecture topology. Indeed, interfacing with increasingly computationally intensive models, such as COSMO, leads to the need of determining the right HW architecture and its configuration to run them under required performances and budget limits. This HW dimensioning problem represents a challenging and complex task for many companies that need assistance from AI experts. In this direction, the challenge could be tackled by integrating domain knowledge held by experts (time constraints, solution quality, budget limits) with data-driven models that learn relationships between HW requirements and algorithm performances, by integrating learning and optimization techniques.

References

- Amiri, M., Mohammad-Khanli, L., 2017. Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications* 82, 93–113.
- Bartolini, A., Beneventi, F., Borghesi, A., Cesarini, D., Libri, A., Benini, L., Cavazzoni, C., 2019. Paving the way toward energy-aware and automated datacentre, in: *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, pp. 1–8.
- Bengio, Y., Goodfellow, I., Courville, A., 2017. *Deep learning*. volume 1. MIT press Massachusetts, USA.
- Bergstra, J., Yamins, D., Cox, D., 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: *International conference on machine learning*, pp. 115–123.
- Borghesi, A., Baldo, F., Lombardi, M., Milano, M., 2020. Injective domain knowledge in neural networks for transprecision computing, in: *International Conference on Machine Learning, Optimization, and Data Science*, Springer. pp. 587–600.
- Borghesi, A., Bartolini, A., Milano, M., Benini, L., 2019. Pricing schemes for energy-efficient hpc systems: Design and exploration. *The International Journal of High Performance Computing Applications* 33, 716–734.
- Breiman, L., 2001. Random forests. *Machine learning* 45, 5–32.
- Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A., 1984. *Classification and regression trees*. CRC press.
- Chatterjee, S., Hadi, A.S., 2013. *Regression analysis by example*. John Wiley & Sons.
- CINECA, 2015. Galileo. <https://www.hpc.cineca.it/hardware/galileo>. Accessed: 2020-11-13.
- De Filippo, A., Borghesi, A., 2022. Constrained hardware dimensioning for ai algorithms, in: *PAIS 2022*. IOS Press, pp. 145–148.
- De Filippo, A., Borghesi, A., Boscarino, A., Milano, M., 2022. Hada: An automated tool for hardware dimensioning of ai applications. *Knowledge-Based Systems* 251, 109199.
- De Filippo, A., Lombardi, M., Milano, M., 2019. How to tame your anticipatory algorithm, in: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 1071–1077.
- Friedman, J.H., 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.
- Fuhrer, O., Osuna, C., Lapillonne, X., Gysi, T., Cumming, B., Bianco, M., Arteaga, A., Schulthess, T.C., 2014. Towards a performance portable, architecture agnostic implementation strategy for weather and climate models. *Supercomputing frontiers and innovations* 1, 45–62.
- Galleguillos, C., Sirbu, A., Kiziltan, Z., Babaoglu, O., Borghesi, A., Bridi, T., 2017. Data-driven job dispatching in hpc systems, in: *International Workshop on Machine Learning, Optimization, and Big Data*, Springer. pp. 449–461.
- Holton, J.R., 1973. An introduction to dynamic meteorology. *American Journal of Physics* 41, 752–754.
- Hood, R., Jin, H., Mehrotra, P., Chang, J., Djomehri, J., Gavali, S., Jespersen, D., Taylor, K., Biswas, R., 2010. Performance impact of resource contention in multicore systems, in: *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, IEEE. pp. 1–12.
- Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K., 2006. Performance prediction and automated tuning of randomized and parametric algorithms, in: *International Conference on Principles and Practice of Constraint Programming*, Springer. pp. 213–228.
- Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K., 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206, 79–111.
- Miu, T., Missier, P., 2012. Predicting the execution time of workflow activities based on their input features, in: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, IEEE. pp. 64–72.
- for Small-scale Modeling, C., a. Cosmo model. URL: <http://www.cosmo-model.org/>. accessed: 2021-01-13.
- for Small-scale Modeling, C., b. General description: General description of the cosmo-model. URL: <http://cosmo-model.org/content/model/general/default.htm>. accessed: 2021-01-13.
- Nadeem, F., Alghazzawi, D., Mashat, A., Fakeeh, K., Almalaise, A., Hargras, H., 2017. Modeling and predicting execution time of scientific workflows in the grid using radial basis function neural network. *Cluster Computing* 20, 2805–2819.
- Paszke, A., Gross, S., et al., 2019. Pytorch: An imperative style, high-performance deep learning library, in: *Wallach, H., Larochelle, H., et al. (Eds.), Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, 2825–2830.
- Pittino, F., Bonfà, P., Bartolini, A., Affinito, F., Benini, L., Cavazzoni, C., 2019. Prediction of time-to-solution in material science simulations using deep learning, in: *Proceedings of the Platform for Advanced Scientific Computing Conference*, pp. 1–9.
- Schulthess, T.C., Bauer, P., Wedi, N., Fuhrer, O., Hoefler, T., Schär, C., 2018. Reflecting on the goal and baseline for exascale computing: a roadmap based on weather and climate simulations. *Computing in Science & Engineering* 21, 30–41.

- Steppeler, J., Doms, G., Schättler, U., Bitzer, H., Gassmann, A., Damrath, U., Gregoric, G., 2003. Meso-gamma scale forecasts using the nonhydrostatic model Im. *Meteorology and atmospheric Physics* 82, 75–96.
- Yoo, A.B., Jette, M.A., Grondona, M., 2003. Slurm: Simple linux utility for resource management, in: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (Eds.), *Job Scheduling Strategies for Parallel Processing*, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 44–60.