POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Dynamic Provisioning and Run-time Optimization of Cloud Workloads

Supervisor
prof. Fulvio Risso

Candidate
Jacopo Marino

Company supervisor
PUNCH Torino S.p.A.
dott. Mauro Bighi

October 2022

Ai miei genitori

Summary

Cloud computing has become very important nowadays for companies, and many of them started offloading job computations instead of increasing the on-premise capacity: the dispatch of those jobs is usually done manually by users, leaving them the choice of the instance and provider to be used. The scope of this thesis is to analyze possible improvements given by the introduction of machine learning in the decision process: the idea was to create a new unit, independent from the scheduler already implemented in the company, having the possibility to extend it to every system deployed. The user is still the author of the choice because he/she is given more information to improve the decision and is not bypassed.

The designed system consists of 2 internal predictors, so it assumes the name of two-stage predictor. Each stage exploits machine learning, but instead of using just one single algorithm, it uses several algorithms to achieve better performance: each one is independently tuned with a set of hyperparameters, with the training on available data as the following step. After the training phase, the best performing algorithm is selected and used for that predictor, in a way that each predictor is independent of the algorithm that is used. The system was tested on 2 projects, considering the combination of two features: data augmentation and Continuous Machine Learning (CML).

The analysis conducted shows two important and different outcomes, that are different but bring to the same final considerations. The first one shows how the system can be wrong if not constantly trained when new data is available, leading to a higher cost with respect to the optimal solution. The second one shows that the system, if well trained and updated, can follow the evolution over the time and learn from data, leading to a potential saving of about 10%. Different results could be obtained under different initial conditions.

As concluding remark, the introduction of machine learning to the job dispatch problem can be effective and lead to optimal solutions only under some conditions. The advantage is the total cost reduction if the current dispatching is not optimal or validate the latter if there is no reduction. The drawback is that the system must be kept updated to follow the evolution of data through time, otherwise, prediction can become inaccurate and lead to much higher costs than expected: the conditions are the number of data available and the application of CML.

Contents

Chapter 1

Introduction

In the last years, cloud computing started to be used by users and companies to improve their flexibility or their workflows. This paradigm became very important in companies businesses, allowing access to top-notch hardware paid only for the actual usage; these factors are relevant both for small companies and for bigger ones: the first can access hardware avoiding upfront costs that could not be affordable, while the second can increase the usage percentage by paying resource only when allocated or needed, saving money on the long run.

Some companies use cloud computing for running HPC simulations, leveraging cloud providers' performance: users prepare models on their personal computers (such as laptops for example), upload them to be run on the cloud and analyze the results when ready, downloading, or working on them on a remote desktop deployed in the cloud.

In order to manage the emerging needs of big companies, some open-source projects and some products appeared on the market to help this kind of computation management: most of these products use several FIFO queues that are utilized to schedule simulations on cloud providers' instances. In this situation, the choice is left to users and there is no entity between job submission and queue choice: the scheduling is done simply by following the order of jobs submission without considering simulations details or users' needs (such as lower cost or lower running time).

1.1 Objectives

The objective of this thesis is not to create a smart scheduler from scratch, but to start from the ones already on the market and create a sort of standalone preprocessing unit independent from the actual queue manager, to be integrated seamlessly with almost all systems, helping users decide which queue is the most suitable for their needs and then proceed the usual way. This unit should be able to predict the execution time of a simulation: it should exploit previous data to gain knowledge and improve its skills continuously following step by step the evolution of the simulation set.

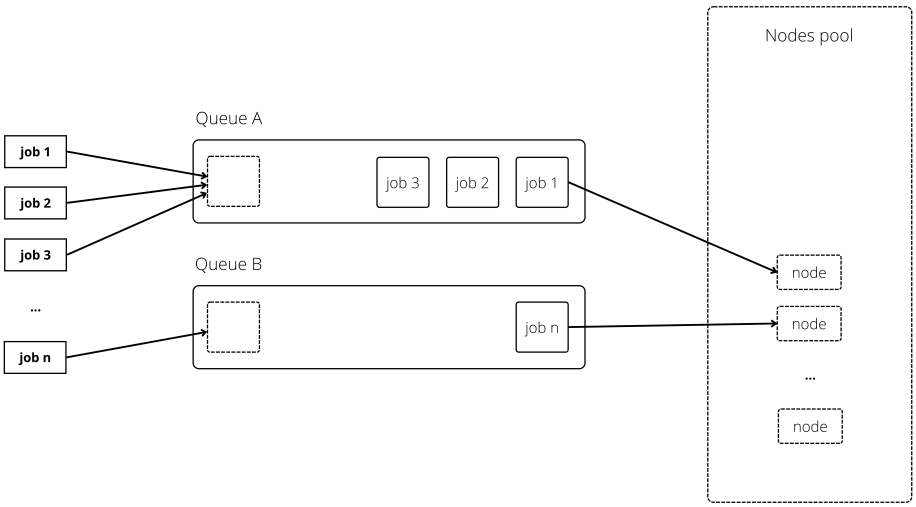

Figure 1.1: FIFO queues for job dispatching

The implementation of this prediction unit should use the cutting-edge technologies offered by the current level of technology: machine learning. This will allow the model to learn on the job, with training on past data and on newer ones when available.

1.2 Outline

- **Chapter 2**: introduction to cloud computing and deployment strategies (public, private, hybrid, and multi), analyzing costs of instances used in the following chapters.
- **Chapter 3**: introduction to the machine learning terminology, with a focus on algorithms used in Chapter 7.
- **Chapter 4**: a brief description of the company, with a focus on its current cloud infrastructure and queue manager used.
- **Chapter 5**: previous work related to a similar application field, using machine learning for cloud execution time prediction.
- **Chapter 6**: a high-level description of the system, focusing on its architecture and functionalities.
- **Chapter 7**: system implementation and choices made to make it suitable for HPC application.
- **Chapter 8**: results of the system, analysis of benefits and drawbacks, with and without data augmentation and continuous machine learning.

- **Chapter 9**: conclusions about the thesis work, with a brief discussion on the possible improvements.

Chapter 2

Introduction to cloud computing

Cloud computing is the access, via the internet, to computing resources, data storage, virtual or private servers, software platforms, etc., hosted in a remote data center managed by a cloud provider. Cloud providers offer these resources with different billing options, to let users decide what is the best option to go for. Compared to the traditional on-premise solution, this paradigm allows users and companies to get powerful resources in a very short time, without waiting for hardware to be bought and set up locally: this can help in saving time and money for the purchase, installation, configuration, management and maintenance of an on-premise solution. These considerations lead to agility and scalability of the cloud: companies that need to scale up the performance of the system can simply add more instances and pay for them, taking also advantage of cloud provider regions deploying resources closer to the users to reduce latency [1].

Cloud computing is based on virtualized IT infrastructure, which includes servers, operating systems, networking, and so on: providers can offer dedicated hardware to users or a portion of that using VMs to isolate users and avoid data leaks between them: for example, a single physical server can be divided in n VMs and each of them sold to a different user. Virtualization allows cloud providers to maximize the system resource usage, selling unused server capacity to other customers. Not surprisingly, the virtualization approach has been adopted also by companies to manage the on-premise infrastructure, so that the maximization approach can be used there as well, avoiding waste of resources: this again improves the agility of the system to adapt to resource utilization [1].

2.1 Public cloud

Public cloud is a type of cloud computing in which cloud providers offer computing resources to users over the public internet: these resources can be free (free-tier) or billed according to pay-per-use or monthly subscription models. Cloud providers own, manage, and have all responsibilities for data centers and the deployed hardware: users use resources, without worrying about malfunctions or service interruptions. Public cloud is a multi-tenant environment: this means that the cloud provider's data centers are shared by all public cloud customers. In the case of leading public cloud providers, Amazon Web Services (AWS), Google Cloud Platform

(GCP), Microsoft Azure, and Oracle Cloud Infrastructure (OCI), those customers can number in the millions. Many companies are moving portions of their computing infrastructure to the public cloud, getting agility and flexibility: this means the possibility of scaling up and down following the company's current needs, reducing cost since customers pay only for what they use [1].

2.2 Private cloud

Private cloud is a cloud environment in which all resources are accessible by one customer only. This solution combines the benefits of cloud computing (scalability, flexibility, etc.) but keeps access control, security, and resource choice of the on-premise solution. A private cloud solution is typically hosted on-premise in the customer's data center, but it can be hosted on a cloud provider infrastructure or built on the rented infrastructure of an off-site data center. Many companies choose private cloud over public cloud because is an easier way (or the only way) to meet their regulatory compliance requirements. Other companies choose this model because they have to deal with confidentiality, intellectual property, sensitive data, etc. (e.g. GDPR in Europe). By building private cloud architecture according to cloud-native principles, an organization gives itself the flexibility and agility to off-load workloads to public cloud or run them in a hybrid cloud environment [1].

2.3 Hybrid cloud

Hybrid cloud is a combination of public and private cloud environments. This solution connects the company's private cloud and the public cloud services into a single flexible infrastructure, in which there is a level of orchestration between resources to use them seamlessly and move workloads between the two clouds. This solution can be used to manage peak loads, using the private cloud capacity during normal behavior and off-loading some computing resources only when needed [1].

2.4 Multi-cloud

Multi-cloud is the use of two or more clouds from two or more different cloud providers, using resources from all based on needs and/or providers' offers. This solution can be integrated with the hybrid cloud, so it becomes a hybrid multi-cloud, which uses two or more cloud providers and the private cloud infrastructure. Multi-cloud solution helps the company to avoid vendor lock-in, so if a provider drops a service it is possible to migrate to another provider and avoid business interruption. Multi-cloud is managed through platforms created for this purpose, providing visibility across multiple cloud providers with a central dashboard in which it is possible to check which nodes and clusters are up and running, plus other parameters [1].

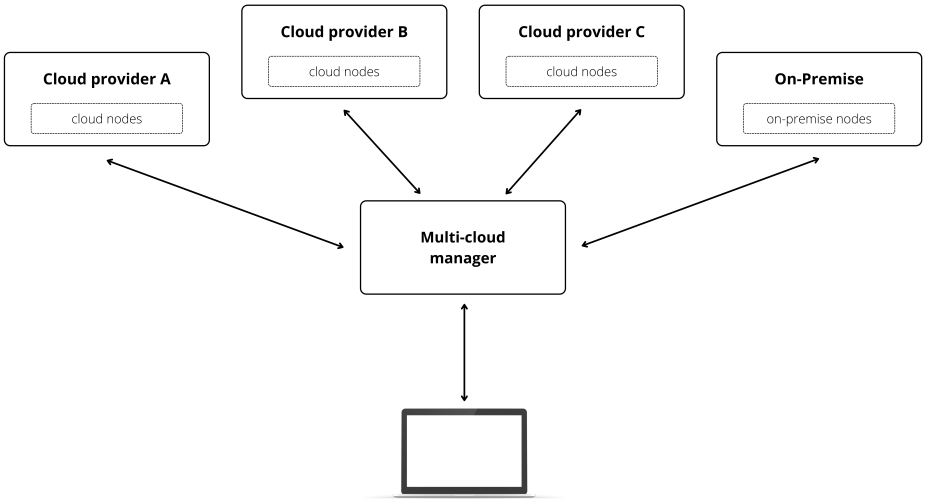

Figure 2.1: Hybrid multi-cloud example with 3 public clouds

2.5 Cloud costs

Cloud providers have different types of billing, depending on the instances' usage and their SLAs (Service Level Agreements): pay-per-use, preemptible, and reserved [2].

Pay-per-use Users are charged a quota directly related to their use of resources, so based on the time usage; there is a minimum time interval equal to 1 minute, after which the billing will be per-second based.

$$c_{instance} = p_{instance} \cdot min\{1\ minute, t_{allocation}\} \tag{2.1}$$

Preemptible Users are charged the same way as the previous type, but in this case, the instances can be preempted by the cloud provider if needed: the advantage is a big discount on instance costs, and the disadvantage is that they can be used only by workloads that can handle interruptions, such as batch processes.

$$c_{instance} = (1 - \Delta) \cdot p_{instance} \cdot min\{1\ minute, t_{allocation}\} \tag{2.2}$$

where Δ is the discount applied on this billing method.

Reserved Users are charged for a period of time (that depends on their choices) and they can have an upfront cost (total or partial), monthly billing, or a combination of the two. The total cost for the instance will be the sum of the monthly billing, plus the upfront cost if present. Some providers can offer a discount if this method is chosen.

$$c_{instance} = p_{monthly} \cdot \#months + p_{upfront} \tag{2.3}$$

2.5.1 Virtual Machine (VM)

Cloud providers offer a set of instances based on the resources that they have: a particular type of Virtual Machine is also called a *VM shape*. Usually, VM shapes are provided with a fixed set of resources, but providers can offer options to get VM more suitable for users' needs, giving them the opportunity the choose the amount of vCPUs and RAM: they are called *flexible VMs*.

Instance Size	vCPU	Memory (GiB)
c6a.large	2	4
c6a.xlarge	4	8
c6a.2xlarge	8	16
c6a.4xlarge	16	32
c6a.8xlarge	32	64
c6a.12xlarge	48	96
c6a.16xlarge	64	128
c6a.24xlarge	96	192
c6a.32xlarge	128	256
c6a.48xlarge	192	384

Table 2.1: Example of AWS VM fixed shapes [3]

The total pay-per-use cost for a fixed shape VM can be computed by multiplying the instance unit cost by the allocation time.

$$c_{VM} = p_{VM} \cdot min\{1\ minute, t_{allocation}\} \tag{2.4}$$

where p_{VM} is the unit price of the chosen VM instance. In the case of flexible VMs, both vCPUs and RAM must be considered in the computation.

$$c_{VM} = (p_{vCPU} \cdot \#vCPUs + p_{RAM} \cdot size_{RAM}) \cdot min\{1\ minute, t_{allocation}\} \tag{2.5}$$

where p_{vCPU} is the unit cost of a virtual CPU (vCPU) and the p_{RAM} is the unit cost of the RAM.

2.5.2 Bare Metal (BM)

Bare Metal instances provide more performance because users can rely on hardware instead of a virtualized environment: all instance resources (CPU, RAM, etc.) belong to a single user. This solution can offer better performance in some cases, so some providers offer this kind of instance. The price model is similar to VMs: pay-per-use or reserved; in the first case, the cost can be computed by multiplying the instance unit cost by the allocation time.

$$c_{BM} = p_{BM} \cdot min\{1\ minute, t_{allocation}\} \tag{2.6}$$

Instance Shape	**OCPU**	**Memory (GB)**
BM.Standard2.52	52	768
BM.Standard.E2.64	64	512
BM.Standard.E3.128	128	2048
BM.Optimized3.36	36	512

Table 2.2: Example of Oracle BM shapes [4]

Chapter 3

Introduction to machine learning

3.1 Regression algorithm

Regression analysis is a collection of statistical methods used in statistical modeling to estimate the relationships between a dependent variable (commonly referred to as the *outcome* or *response* variable, or a *label* in machine learning terminology) and one or more independent variables (often called *predictors* or *features*). The most frequent type of regression analysis is linear regression, in which the line (or a more sophisticated linear combination) that best fits the data according to a certain mathematical criterion is found. The method of ordinary least squares, for example, computes the unique line (or hyperplane) that minimizes the sum of squared differences between the true data and that line (or hyperplane) [5].

A regression algorithm's output is a continuous value, which means that it will provide a real number starting from one or more inputs. A model can be described as a function, whose parameters are tuned during the training phase, analyzing available data. Regression models involve the following components [5]:

- the unknown parameters, often denoted as a scalar or vector β
- the independent variables, which are observed in data and are often denoted as a vector $\mathbf{X_i}$ (where i denotes a row of data)
- the dependent variable, which are observed in data and often denoted using the scalar $\mathbf{Y_i}$
- the error terms, which are not directly observed in data and are often denoted using the scalar e_i

Most regression models propose that $\mathbf{Y_i}$ is a function of $\mathbf{X_i}$ and β, with e_i representing an additive error term that may stand in for un-modeled determinants of $\mathbf{Y_i}$ or random statistical noise [5]:

$$\mathbf{Y_i} = f(\mathbf{X_i}, \beta) + e_i \tag{3.1}$$

3.2 Machine Learning algorithms

The machine learning task of learning a function that maps an input to an output based on example input-output pairs is known as *supervised learning*. It derives a function from labeled training data, which consists of a set of training examples. Each example in supervised learning is a pair consisting of an input object (usually a vector) and the desired output value (also called the supervisory signal). A supervised learning algorithm examines the training data and generates an inferred function that can be used to map new examples: in an ideal scenario, the algorithm will be able to accurately determine the class labels for unseen data; this requires the learning algorithm to be able to generalize from the training data to unseen situations in a *reasonable* way. The statistical quality of an algorithm is measured through the so-called generalization error: a measure of how accurately an algorithm is able to predict output values for previously unseen data [6].

3.2.1 Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron is a fully connected class of feedforward artificial neural networks (ANN). An MLP is composed of at least three layers of nodes: the input layer, one or more hidden layers, and the output layer. Each node of hidden and output layers is a neuron that implements a non-linear function. The MLP uses a supervised learning approach called *backpropagation* for the training phase. The MLP and a linear regression differ because the former has multiple layers and non-linear activation. It can distinguish data that is not linearly separable [7].

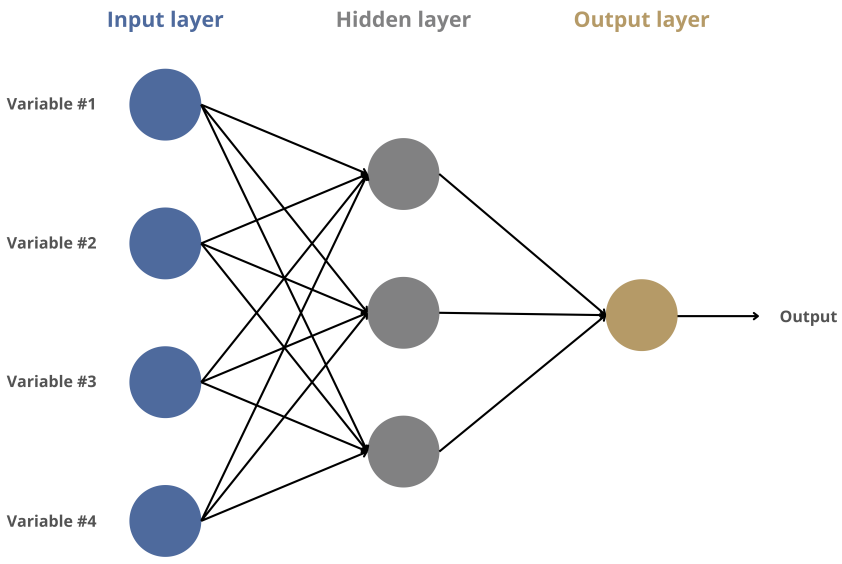

Figure 3.1: MLP network example with one hidden layer

3.2.2 Support-Vector Machine (SVR)

Support Vector Machines (also support-vector networks) are supervised learning models, in which learning algorithms analyze data for classification and regression problems. Given a set of training examples, each one labeled with one of two categories, an SVM training algorithm builds a model to assign new examples to one category or to the other, making it a non-probabilistic binary linear classifier. SVM maps training examples to points in space, to maximize the gap between the two categories: new examples are mapped in the same space and predicted with a label (that identifies a category) based on which side of the gap they fall. In addition to linear classification, SVM can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces [8].

3.2.3 Linear Support-Vector Machine (LSVR)

The Linear Support-Vector Machine works in the same way as an SVM, using a linear kernel for training and predictions [8].

3.2.4 Random Forest (RF)

The Random Forest (or random decision forest) is an ensemble method for classification, regression, and other tasks that consists in building a multitude of decision trees during the training phase. Ensemble methods use multiple learning algorithms to obtain better performance in the prediction than using any single constituent algorithm alone. For the classification task, the output of a random forest is the class selected by the majority of trees, while for the regression task it is the average prediction of every single tree. The advantage of random forests with respect to decision trees is the trend of the latter to overfit over training data: generally random forests outperform decision trees, but data characteristics can affect their performance [9].

Random forests are frequently used as *blackbox* models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration [9].

3.2.5 Decision Tree (DT)

Decision Tree is a supervised learning approach used in statistics, data mining, and machine learning. A classification or regression decision tree is used as a predictive model to draw conclusions about a set of observations. Decision trees where the target value belongs to a discrete set of values are called classification trees: in these trees, the leaves represent the class labels, while branches represent the conjunctions of features that lead to the class label. Decision trees where the target value can assume continuous values (usually real numbers) are called regression trees [10].

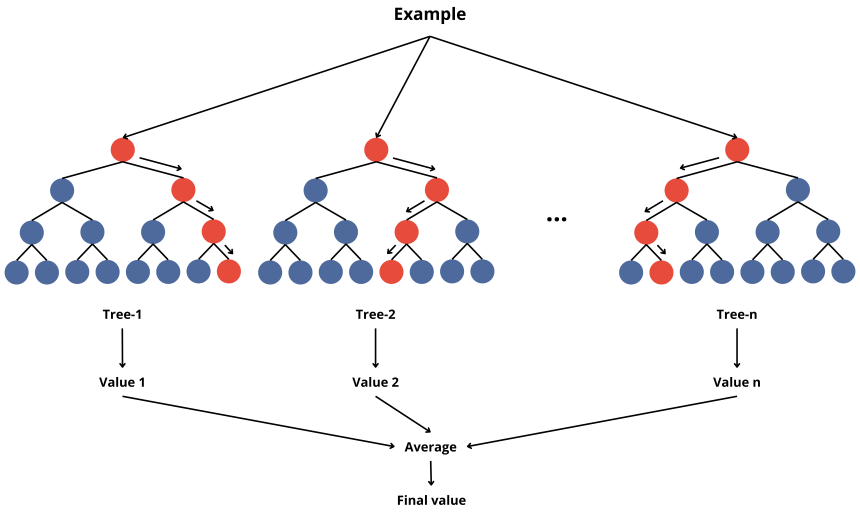

Figure 3.2: Random Forest example with n estimators

Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity. In decision analysis, a decision tree can be used to explicitly visualize decisions and decision-making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision-making) [10].

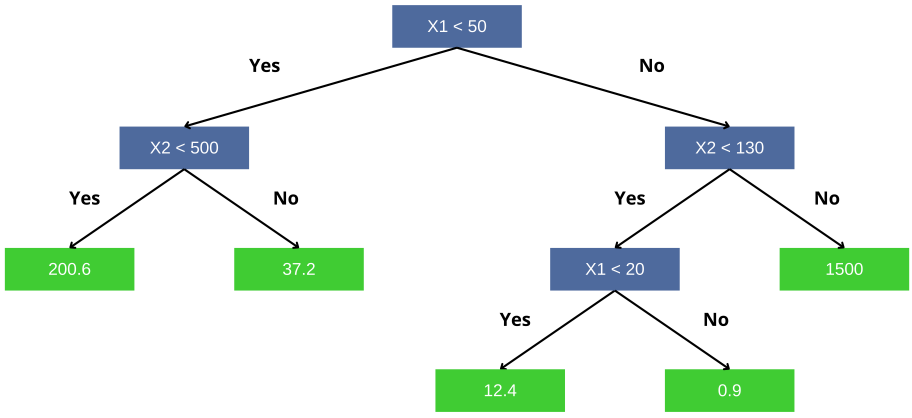

Figure 3.3: Decision Tree example with two features

3.2.6 Linear Regression (LR)

Linear regression is a linear approach for modeling the relationship between a dependent variable (the output) and one or more independent variables (the inputs). The case of one independent variable is called simple linear regression; for more

than one, it is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable. In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data: these models are called linear models [11].

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the *lack of fit* in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares cost function as in ridge regression (L^2 - norm penalty) and lasso (L^1 - norm penalty). Conversely, the least squares approach can be used to fit models that are not linear models. Thus, although the terms *least squares* and *linear model* are closely linked, they are not synonymous [11].

In linear regression, the model states that the dependent variable, y_i is a linear combination of the parameters (but need not be linear in the independent variables). For example, in simple linear regression for modeling n data points there is one independent variable: x_i, and two parameters, β_0 and β_1. A straight line would be described as follow [11]:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \quad i = 1, \ldots, n \tag{3.2}$$

In multiple linear regression, there are several independent variables or functions of independent variables [11].

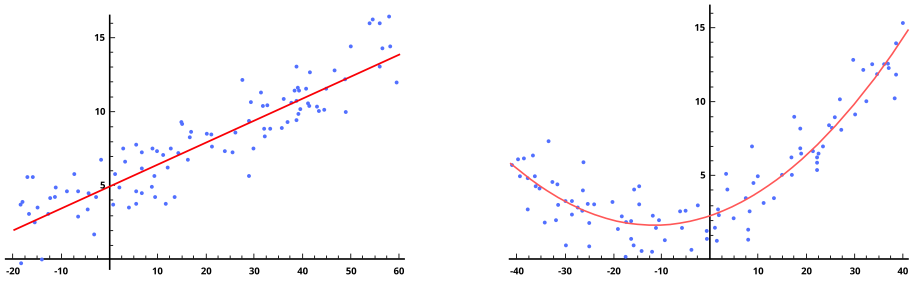

Figure 3.4: Linear (left) vs Polynomial (right) Regression example

3.2.7 Polynomial Regression (PR)

Polynomial regression is a type of regression in which the relationship between the independent variable x and the dependent variable y is modeled as an nth-degree polynomial in x. Polynomial regression fits a nonlinear relationship between the input x and the corresponding conditional mean of y, denoted as $E(y|x)$. Although polynomial regression fits a nonlinear model to the data, the estimation problem is linear, which means that the regression function $E(y|x)$ is linear in the unknown parameters that are estimated from the data: polynomial regression is considered to be a special case of multiple linear regression. It uses the same linear regression of subsection 3.2.6, but before training the model, the input variables are scaled from a linear domain to a polynomial one, creating inputs (power of one input or multiplication between inputs) [12].

For example, adding a term in x_i^2 to the formula 3.2 gives a parabola:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i, \quad i = 1, \ldots, n \tag{3.3}$$

This is still linear regression, although the expression on the right hand side is quadratic in the independent variable x_i, it is linear in the parameters β_0, β_1 and β_2 [12].

3.2.8 K-Nearest Neighbors (KN)

The K-Nearest Neighbors algorithm (k-NN) is a non-parametric supervised learning method used for classification and regression. In both cases, the input consists of the k closest training examples in a data set, while the output depends on whether k-NN is used for classification or regression [13]:

- In classification, the output is a class label. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor [13].
- In regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors [13].

k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically[13].

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor. The neighbors are taken from a set of objects for which the class (for classification) or the object property value (for regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required [13].

3.2.9 Mean Value (MV)

This is not a real regression algorithm, anyway the output is a continuous value. This algorithm is composed of 5 different sub-algorithms: *arithmetic mean*, *geometric mean*, *harmonic mean*, *mode*, and *median*.

Arithmetic mean The arithmetic mean is the sum of all values divided by the count of them.

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \ldots + x_n}{n} \tag{3.4}$$

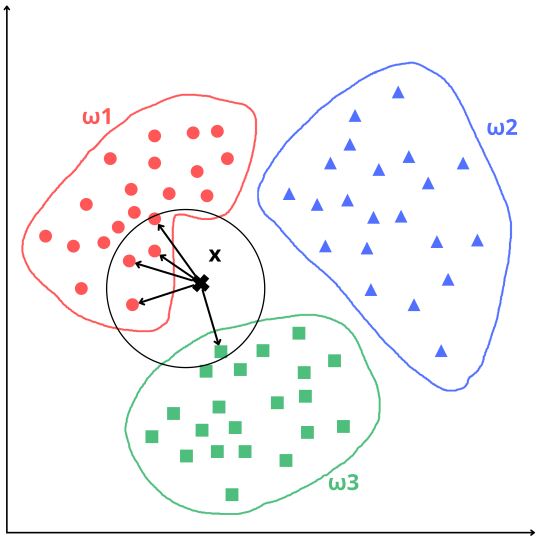

Figure 3.5: K-Nearest Neighbors example with 3 clusters [14]

Geometric mean The geometric mean is the n-th root of the product of all values.

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \dots x_n} \tag{3.5}$$

Harmonic mean The harmonic mean is the reciprocal of the arithmetic mean of the reciprocals of all values.

$$\left(\frac{1}{n}\sum_{i=1}^{n} x_i^{-1}\right)^{-1} = \left(\frac{x_1^{-1} + x_2^{-1} + \dots + x_n^{-1}}{n}\right)^{-1} \tag{3.6}$$

Mode The mode is the value that appears most often in a set of data values [15]. Suppose a set $x = \{x_1, x_1, x_1, x_2, x_3\}$: the mode is the value x_1, because is the one with the highest frequency in the set.

Median The median is the value separating the higher half from the lower half of a data sample [16].

Each sub-algorithm is used and then all performance results are compared, so the best one is chosen as the resulting algorithm to be used for predictions.

Chapter 4

HPC at PUNCH Torino S.p.A.

4.1 Company introduction

PUNCH Torino S.p.A. is the Core Company that allows the PUNCH Group to lead the engineering of innovative propulsion systems and control solutions, based on the unique combined expertise of developing, producing, and integrating proven technologies, systems, and processes towards turnkey solutions [17].

The company utilizes a hybrid cloud infrastructure to run simulations used in the development process of an engine, offloading simulations to different cloud providers and retrieving the results once ready or working on them on remote desktops deployed on the cloud.

4.2 Current infrastructure

The current infrastructure is a hybrid cloud, composed of computational nodes on-premise alongside on-demand instances deployed on three cloud providers: Oracle, Google, and Azure. The management of scheduling is demanded to a commercial product developed by Altair (PBS Professional), introduced in the subsection 4.2.1: the users prepare the input files for the simulations, upload them on the web interface of the scheduler, choose which instances they would use, and the system will dispatch the simulations on the correct queues to satisfy the requirements. Once the simulations finish, the output data is moved from the instances' temporary storage to the warm storage of the cloud provider to which the instances belong. The scheduler will continue the dispatching until queues are empty or the maximum number of parallel instances has been reached and monitors when a new simulation can be submitted to a cloud or on-premise instance.

4.2.1 Altair PBS Professional

Altair PBS Professional is an industry-leading workload manager and job scheduler for HPC and High-throughput Computing. PBS Professional is a fast, powerful

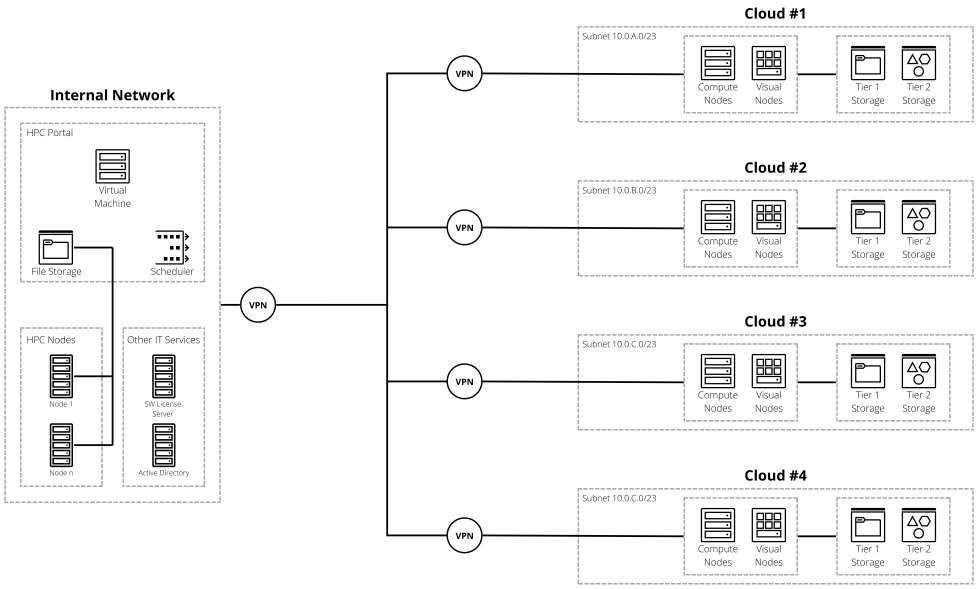

Figure 4.1: Current infrastructure [18]

workload manager designed to improve productivity, optimize utilization and efficiency, and simplify administration for clusters, clouds, and supercomputers - from the biggest HPC workloads to millions of small, high-throughput jobs. PBS Professional automates job scheduling, management, monitoring, and reporting, and it is the trusted solution for complex Top500 systems as well as smaller clusters [19].

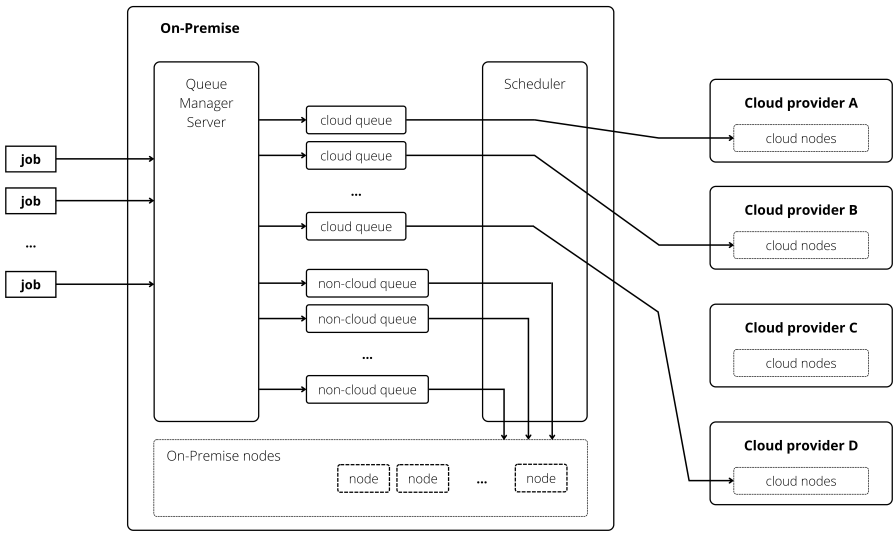

Figure 4.2: Simplified version of a queue manager [20]

4.2.2 Cloud providers

The current infrastructure uses three cloud providers (Azure, Google, and Oracle), plus the on-premise computational units. For each provider, an instance has been identified to be used, in particular, the choice was made upon HPC instances, that offer better performance for some types of computations, plus the InfiniBand connectivity in some cases, to allow the possibility of scaling on more instances.

4.2.3 Instances

The instances chosen are provided by cloud providers for the same purpose: HPC computing. These instances have some peculiarities with respect to *standard* ones because they are optimized for HPC computations and some of them offer the InfiniBand connectivity with the RDMA over it: Oracle and Azure are actually offering this feature, while Google is not and this will lead to different uses for different platforms because the scalability is affected. Table 4.1 reports the 3 types of instances implemented in the current infrastructure.

Provider	Instance	#Cores	RAM (GB)
Oracle	BM.Optimized3.36	36	512
Google	c2-standard-60	30	240
Azure	HB120rs v3	120	448

Table 4.1: Chosen instances by the company

The Oracle instance is a Bare Metal because it offers slightly better performance with respect to VM, and it is the only one that offers the InfiniBand connectivity, so considering the possibility of scaling, it is the only one suitable for the infrastructure [4].

The Azure instance is quite different from the other 2 because it is a VM that offers 120 cores, with the InfiniBand support and so it can scale as the Oracle one. The cloud provider offers also smaller shapes, but they are based on this VM and it is the same with just the other cores disabled and the same hourly price, so the reference instance is this one, even if the system will use a smaller shape [21].

The Google instance is similar in the number of cores to the Oracle one: it exposes 60 vCPUs with hyper-threading, but this functionality is disabled to get some more performance using only the physical cores, resulting in a VM with 30 cores [22]. This instance does not offer the InfiniBand connectivity, so the scalability is possible in theory, but in reality, the performance degrades with more than one instance.

Given the details about instances on cloud providers, it is possible to see in Table 4.2 the possible scenarios, composed of one or more instances, or even by a fraction of it (like in the case of Azure).

The Google case considers also scalability, which will be considered later in the thesis, and for this reason, it is reported here, but in reality, it should not be used.

Provider	Instance	#Instances	#Cores	RAM (GB)
Oracle	BM.Optimized3.36	1	36	512
Oracle	BM.Optimized3.36	2	72	1024
Oracle	BM.Optimized3.36	3	108	1536
Oracle	BM.Optimized3.36	4	144	2048
Google	c2-standard-60	1	30	240
Google	c2-standard-60	2	60	480
Google	c2-standard-60	3	90	720
Google	c2-standard-60	4	120	960
Azure	HB120rs v3	1	32	448
Azure	HB120rs v3	1	64	448
Azure	HB120rs v3	1	96	448
Azure	HB120rs v3	1	120	448

Table 4.2: Instances used by the company

InfiniBand and RDMA

The use of RDMA is necessary to get the best performance out of more instances, otherwise, the results will be just a slower and more expensive solution.

InfiniBand (IB) InfiniBand is a high-performance computing networking communications technology with extremely high throughput and low latency. It is used for data interconnection both among and within computers. InfiniBand is also utilized as a direct or switched connectivity between servers and storage systems, and also between the latter. It is scalable and has a switched fabric network topology [23].

RDMA Remote direct memory access (RDMA) is direct memory access from one computer's memory to that of another without involving either computer's operating system. This allows for high-throughput, and low-latency networking, which is particularly beneficial in massively parallel computer clusters. RDMA enables zero-copy networking by allowing the network adapter to move data from the wire directly to application memory or from application memory directly to the wire, removing the need for the operating system to copy data between application memory and the data buffers. Such transfers do not require any effort from CPUs, caches, or context switches, and they run in parallel with other system tasks: this reduces latency in message transfer [24].

Chapter 5

Related work

This thesis work started from the paper *Predicting Workflow Task Execution Time in the Cloud Using A Two-Stage Machine Learning Approach*, in which authors faced a similar problem: the goal was to apply machine learning to the cloud to predict the run time of different tasks with different input data [25]. The abstract of the paper is reported below.

> *Many techniques such as scheduling and resource provisioning rely on performance prediction of workflow tasks for varying input data. However, such estimates are difficult to generate in the cloud. This paper introduces a novel two-stage machine learning approach for predicting workflow task execution times for varying input data in the cloud. In order to achieve high accuracy predictions, our approach relies on parameters reflecting runtime information and two stages of predictions. Empirical results for four real world workflow applications and several commercial cloud providers demonstrate that our approach outperforms existing prediction methods. In our experiments, our approach respectively achieves a best-case and worst-case estimation error of 1.6 and 12.2 percent, while existing methods achieved errors beyond 20 percent (for some cases even over 50 percent) in more than 75 percent of the evaluated workflow tasks. In addition, we show that the models predicted by our approach for a specific cloud can be ported with low effort to new clouds with low errors by requiring only a small number of executions* [25].

5.1 Two-Stage Machine Learning Approach

The paper introduced a two-stage prediction approach, in which the desired output derives from the concatenation of two prediction blocks. The considered input parameters can be divided into two groups: pre-runtime and runtime parameters. *Pre-runtime parameters* can be determined before a task is executed on the cloud; they include the input of a task and the parameters that describe the virtualized environment in which the task will be executed. *Runtime parameters* are determined by executing a task and they reflect the difference between tasks on different

instances and cloud providers. The parameters considered in the paper are reported in Table 5.1 [25].

Pre-Runtime Parameters	Task input data	Input parameters of task
	VM types	Number of vCPUs, Memory capacity
Runtime Parameters	uCPU	CPU used time at user level
	sCPU	CPU used time at system level
	Memory usage	Memory used by task
	Write operations	Number of written blocks of task
	Read operations	Number of read blocks of task
	File transfer	Size of transferred files by task
	Bandwidth	Bandwidth used by task

Table 5.1: Parameters used to model task execution times [25]

The first stage of prediction takes as input the pre-runtime parameters and returns as output the runtime parameters: each item of the output set can be predicted using any state-of-the-art machine learning regression algorithm. Once the output set is ready, the pre-runtime and runtime parameters are used together in the second stage to predict the execution time of the task [25].

The system proposed has been evaluated using different types of tasks from different software, such as Blender. The data, used in the training and the validation phases, was collected by running those example tasks on different cloud providers, using both a single-stage and the two-stage approach, to compare the two solutions and show how the system introduced performs in comparison to older solutions. The results show how the two-stage approach can outperform the single-stage, which usually are based only on pre-runtime parameters to predict the execution time of the task [25].

Chapter 6

System architecture

The goal of the thesis is to create a standalone prediction unit, to be integrated seamlessly with almost all schedulers present on the market, helping users decide which queue is the most suitable for their needs. This unit should be able to predict the execution time (also referred to as the runtime) of a simulation, starting from inputs known a priori. The next step is to use this output to compute the cost of the simulation on different cloud providers and schedule it on the right one to implement a cost-optimization strategy.

The system has been designed to be sitting in between the jobs and the queues, so starting from Fig.1.1, after the implementation the smart queue management assumes the form of Fig.6.1. The job submission phase is still demanded to users, but they are provided with information about the best queue to use, aiming a cost optimization: if they want to use a specific queue, they are able to do it, because the system is not substituting them, but it augments the information available to help in making decisions.

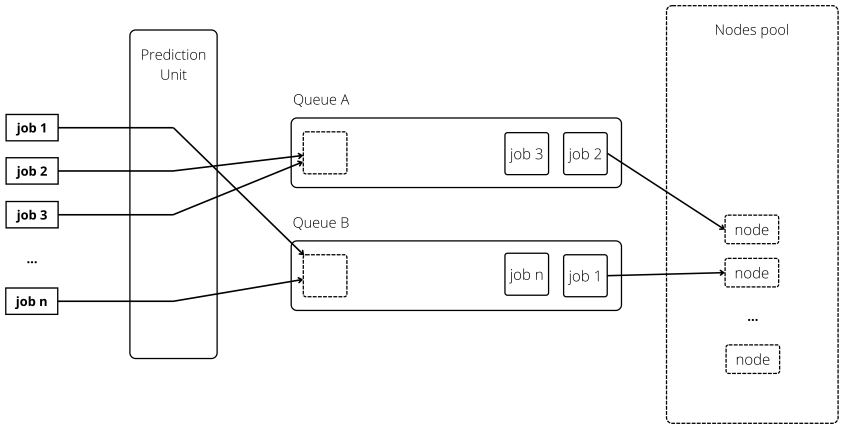

Figure 6.1: Queue management with prediction unit

6.1 Two-stage predictor

The predictor consists of two sub-predictors, introduced by the paper reported in the section 5. The first one predicts the runtime parameters starting from the pre-runtime parameters, while the second one predicts the execution time starting from both pre-runtime and runtime parameters. The runtime parameters predictor is actually a set of predictors, because each runtime parameter has its own predictor, to better manage the differences between the relationships inputs-output.

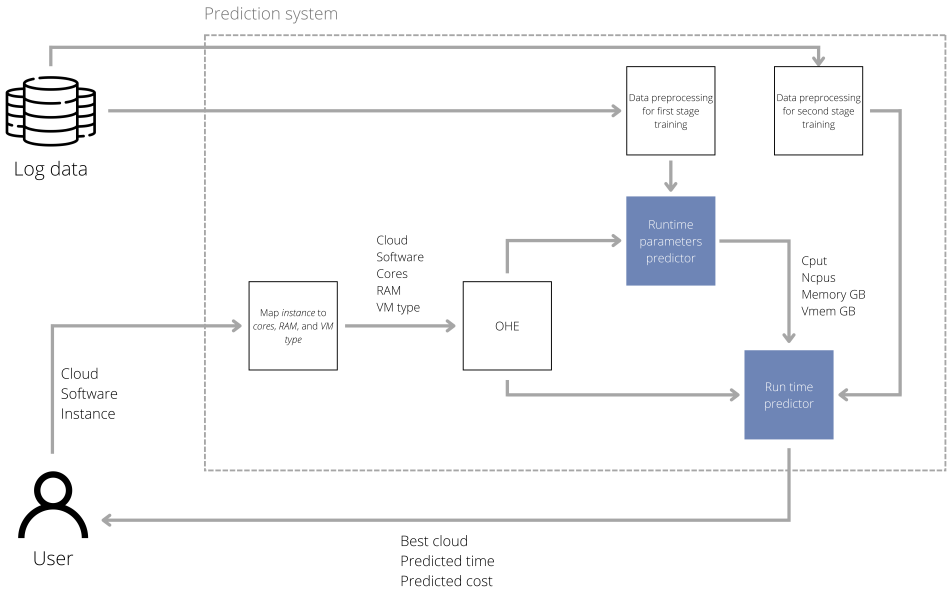

Figure 6.2: System structure overview

Both predictors have the same structure, and they differ only in the inputs and outputs. 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 providing y as result:

$$f : \mathbf{x} \rightarrow y \tag{6.1}$$

The function f is implemented by a regression algorithm because the output y is a continuous value and not a label. Considering this, the internal structure of a predictor will be now discussed. A predictor is composed of 9 regression algorithms (*MLP*, *SVR*, *LSVR*, *RF*, *DT*, *LR*, *PR*, *KN*, *MV*) presented in in Chapter 3. During the training phase, the first step is doing a hyper-parameter search to get the best algorithms' parameters that are able to provide the best predictions; the second step is the actual training of models using the hyper-parameters found, storing the trained ones on disk to have a copy in case of unexpected program termination. Once all models are trained, there is a comparison between the performance of each algorithm, and the best one is marked as the *best* one for that specific output; after this, the predictor is ready to be used.

When a predictor is requested a prediction, it must receive the inputs associated to it in order to satisfy the request. The predictor searches on disk for the best ML model and uses it to predict the value, then returns the output.

6.2 Pre-runtime parameters

Pre-runtime parameters are known a priori, and they are related to the characteristics of the VM and to the software used: they don't need to be predicted and must be known, otherwise, the prediction will not work. They are 5: **cloud**, **software**, **cores**, **ram** and **vm type**.

Cloud The *cloud* input defines which platform the job will run on, and it can assume one of the values in the set c, so a job can run both on cloud providers or on the on-premise infrastructure.

$$c = \{oracle, google, azure, aws, onprem\} \tag{6.2}$$

Software Based on the simulation that has to be done, different software is chosen so that it follows the job needs. The *software* input can assume a value in the set s: there is one software that is repeated two times (*Software2* and *Software2Opt*), but it is used in a different manner, and for this reason, it is considered as standalone for the thesis purpose.

$$s = \{Software1, Software2, Software2Opt, Software3\} \tag{6.3}$$

Cores, ram and vm type The *cores* input depends on the cloud provider, because instances are similar but not identical, so the values should be adapted to follow providers instance shapes. These three inputs are strictly dependent from each other, because shapes are fixed and so values are a tuple and not independent.

$$(cores, ram, vm\ type) \tag{6.4}$$

$$v = \{1, 2, 3, 4\} \tag{6.5}$$

The *vm type* input can assume a value in the set v and it was introduced to standardize the differences between cloud providers because the set of resources on provider A can be full filled using one instance, while on provider B there could be the necessity to spread on multiple instances.

The pre-runtime parameters of an entry can be summarized as a tuple, that can then be fed to the predictors.

$$(cloud \in c, software \in s, cores, ram, vm\ type \in v) \tag{6.6}$$

Cloud	vCPUs	RAM (GB)	VM Type
Oracle	36	512	1
	72	1024	2
	108	1536	3
	144	2048	4
Google	30	240	1
	60	480	2
	90	720	3
	120	960	4
Azure	32	448	1
	64	448	2
	96	448	3
	120	448	4

Table 6.1: Instance shapes details for pre-runtime parameters

6.3 Runtime parameters

Runtime parameters are not known a priori, so their value can be known only when a simulation has terminated and is no more in the running state. These parameters cannot be known, but they can be predicted. They are 4: **cput**, **ncpus**, **memory GB**, **vmem GB**.

Cput The *cput* parameter is the time spent by the CPU to execute the machine instructions of the simulation. This value follows the characteristics of the instance used, because it considers all the vCPUs involved used by the simulation to run and so it is directly proportional to the final output of the system.

Ncpus The *ncpus* parameter represents the number of used vCPUs by a simulation because it could happen that fewer cores are used and so the instance is not fully utilized.

$$1 \leq ncpus \leq cores_{instance} \tag{6.7}$$

Memory GB The *memory GB* parameter is the size of the memory utilized during the simulation run: it is a conversion to Gigabytes because the provided value is in bytes.

$$0GB < memory\ GB \leq RAM_{instance} \tag{6.8}$$

Vmem GB The *vmem GB* parameter is the size of the virtual memory utilized during the simulation run: it is a conversion to Gigabytes because the provided value is in bytes.

$$0GB < vmem\ GB \leq vmem_{instance} \tag{6.9}$$

6.4 Output

The output of the prediction unit is the execution time of the job expressed in seconds, predicted using both the pre-runtime and runtime parameters.

$$\begin{bmatrix} cloud \\ software \\ cores \\ ram \\ vm\ type \\ cput \\ ncpus \\ memory\ GB \\ vmem\ GB \end{bmatrix} \rightarrow time_{predicted} \tag{6.10}$$

The output of the system is not only the execution time by the way, but is the cloud provider with the lowest cost (and the execution time itself). When a job needs a time estimation, the system is fed with n options, where n is the number of cloud providers implemented and usable to deploy compute nodes: given an input tuple t_1 with $cloud_1$, other $n-1$ tuples are created changing the values to be coherent along inputs. The cloud parameter is changed with all the other available. *Cores*, *ram* and *vm type* values are changed accordingly to the table 6.1, to keep the instance shape equal to the one provided by cloud providers, scaling the *ncpus* on the new values so that the ratio $ncpus/cores$ stays constant.

$$\begin{aligned} i_1 &= (cloud_1, \ldots, cores_1, ram_1, \ldots, ncpus_1, \ldots) \\ i_2 &= (cloud_2, \ldots, cores_2, ram_2, \ldots, ncpus_2, \ldots) \\ &\ldots \\ i_n &= (cloud_n, \ldots, cores_n, ram_n, \ldots, ncpus_n, \ldots) \end{aligned} \tag{6.11}$$

The entries of the set $i = \{i_1, i_2, \ldots, i_n\}$ of all these input tuples are fed into the prediction system, to get the execution time for each entry. When the time prediction is available, it is multiplied by the cost of the instance used: these values are then compared and the best one determines which cloud provider is the cheapest for the simulation. The provided values are then the best cloud, the predicted time, and the predicted cost.

$$\mathbf{i} \rightarrow (cloud_{best}, time_{predicted}, cost_{predicted}) \tag{6.12}$$

Chapter 7

Implementation

The system has been implemented using the Python programming language, to take advantage of the available libraries for machine learning and to speed up the development: in particular, the library used to implement the machine learning algorithms is *Scikit-learn* [26]. The implementation started from tests on machine learning algorithms exploiting data augmentation, adding Continuous Machine Learning (CML) as the last step to improve performance in the long run.

7.1 Data available

The data available is the log provided by the PBS Professional queue manager, in which it is possible to find one entry for each job submitted by users: these entries have several fields, and some of them are the one used by the prediction system. The log file can be exported from PBS Professional platform as a CSV file and then parsed in code. The fields of interest are a subset of the present ones because not all of them are useful for the scope of the thesis. Each predictor has its own set of inputs and output, taken from the provided fields: the data submitted is the same for each predictor, but then internally data are processed to put them in the correct form and with the correct format.

The number of entries in a dataset is fundamental for the training phase of a machine learning model, and the available one is about 10K entries. The data set entries have been grouped by *project*, and this step allows to avoid interference between different projects. A *project* describes a type of product and all related details: a product can be very different from the others, so it is better to split data and avoid interference, reducing the noise in predictors and thus getting more accurate results.

7.1.1 Feature extraction

Not all data provided by logs are useful for the system, and for this reason, features have been extracted. *Feature extraction is the process of defining a set of features*

which will most efficiently or meaningfully represent the information that is important for analysis and classification [27]. The set of inputs is different for each sub-predictor because each one uses different inputs to predict a different output.

Consider again the set $c = \{oracle, google, azure, aws, onprem\}$ containing all cloud providers possible: the input variable *cloud* can assume one of those string values. In order to standardize the input set, converting it to a set of numbers, and avoid problems with some models, a transformation must be applied. There are 2 possible ways of doing this: the first is assigning an integer number to each possible value, while the second is to use *One-Hot Encoding (OHE)*; the latter is an encoding method to convert categorical data to values that can be understood by the machine learning model. The variable becomes an array of variables in which each item can assume only 0 or 1 value: this conversion passes from categorical variable to n variables, and one of them is set to 1, while the others are set to 0. n is the cardinality of the set of different values that the variable can assume and in the case of *cloud* input $n = 5$, which is the same of c set.

cloud		**oracle**	**google**	**azure**	**aws**	**onprem**
oracle		1	0	0	0	0
oracle		1	0	0	0	0
google		0	1	0	0	0
azure	→	0	0	1	0	0
azure		0	0	1	0	0
...						
oracle		1	0	0	0	0

Table 7.1: OHE example of *cloud* variable

This encoding has been done for *cloud* and *software* variables: after this, the data is ready to be used for further processing and for training models. Obviously, when new data has to be submitted to the prediction system, the entries must be pre-processed in the same way described until now, including the OHE: this has to be done because the system must be fed data in the same way done during the training phase.

7.2 Data augmentation

Machine learning algorithms need lots of data to be trained in order to get high accuracy: splitting the data set by project, led to some projects with hundreds of examples and to some with only few; this is a problem and so it has to be solved, using existing data to reach around 10K examples for each project. There are two different data augmentation algorithms that are done by the system: the first is for data used to train the pre-runtime parameter predictors, while the second is for data used for the runtime predictor, but both follow the same steps.

- **Step 1**: each entry is replicated n times, depending on the number of cloud providers active and usable, changing the *cloud* column and the related ones,

which are *core*, *ram*, *ncpus*, *cput* and *runtime*, modified using the scalability analysis described in subsection 7.2.1. With this approach, it is possible to map the execution time of a job on different cloud providers allowing the generation of data also for providers never seen by the system, to make the model ready for all prediction requests.

$$
\begin{gathered}
i_1 = (cloud_1, cores_1, ram_1, \ldots, ncpus_1, cput_1, runtime_1) \\
\downarrow \\
i_1 = (cloud_1, cores_1, ram_1, \ldots, ncpus_1, cput_1, runtime_1) \\
i_2 = (cloud_2, cores_2, ram_2, \ldots, ncpus_2, cput_2, runtime_2) \\
\ldots \\
i_n = (cloud_n, cores_n, ram_n, \ldots, ncpus_n, cput_n, runtime_n)
\end{gathered}
$$

- **Step 2**: each entry is replicated j times, multiplying the values for a random coefficient which uses a Δ defined in code and computed as

$$rc = 1 + random(-\Delta, \Delta), \quad 0 \leq \Delta < 1 \tag{7.1}$$

The value j is computed by dividing the desired number of examples by the number of cloud providers and then divided by the number of current examples: the value obtained is converted into an integer, because is not strictly necessary to have exactly the number of desired examples, but a close value. An example for *runtime* column augmentation is the following (the *cput* is also modified because is directly related to *runtime*, and if not implemented could lead to problems during predictions):

$$
\begin{gathered}
i_1 = (\ldots, cput_1, runtime_1) \\
\downarrow \\
i_1 = (\ldots, cput_1, runtime_1) \\
i_2 = (\ldots, cput_1 \cdot rc_2, runtime_1 \cdot rc_2) \\
\ldots \\
i_j = (\ldots, cput_1 \cdot rc_j, runtime_1 \cdot rc_j)
\end{gathered}
$$

- **Step 3**: each entry is replicated 3 times, one for each *vm type* and columns are scaled following this change, similar to step 1, but scaling in the same provider and not between different ones.

$$
\begin{gathered}
i_1 = (cloud_1, cores_1, ram_1, \ldots, ncpus_1, cput_1, runtime_1) \\
\downarrow \\
i_1 = (cloud_1, cores_1, ram_1, \ldots, ncpus_1, cput_1, runtime_1) \\
i_2 = (cloud_1, cores_2, ram_2, \ldots, ncpus_2, cput_2, runtime_2) \\
i_3 = (cloud_1, cores_3, ram_3, \ldots, ncpus_3, cput_3, runtime_3) \\
i_3 = (cloud_1, cores_4, ram_4, \ldots, ncpus_4, cput_4, runtime_4)
\end{gathered}
$$

Step 1 is always present, and step 2 can be enabled or not: the reason is that without step 1 it will be impossible for the system to predict an execution time for a provider never seen. Step 3 has been implemented and tested, but it is always disabled because it adds noise to the predictions, so it is better to avoid scaling instances for the training phase.

7.2.1 Estimation of other clouds performance

Step 1 of data augmentation estimates the execution time on different providers with respect to the current one: this is achieved by using benchmarks on cloud providers with representative simulations, interpolating the scalability curve to obtain models to move between providers instances. These scalability models have been created from the benchmarks (reference simulations) done with the software used, and passed to the latter as inputs: these simulations are not from production, so they cannot leak information about companies, but they are representative and can be assumed similar to production ones. Benchmarks have been run not for all cores configurations, but only for a couple of cases (and sometimes only for one, and more details will be provided on how this situation has been managed).

The estimations are done between different cloud providers keeping the same *vm type*: each type is associated with a score, which is the runtime on that specific platform with that specific resources, with an example shown in Table 7.2. Values reported are anonymized, in order to not disclose sensitive information about real runtimes, even if they are benchmark simulations and so not related to the production environment. Using Table 7.2, it is possible to estimate the performance

VM Type	Oracle	Google	Azure	AWS
1	5287	7034	5660	7442
2	4974	7221	5514	6982
3	4660	7409	5369	6522
4	4346	7596	5259	6061

Table 7.2: Software 1 scores

of a simulation on other cloud providers, by simply taking the ratio of the new and the old scores, multiplying by it the execution time: the result is the estimation of the time on the new cloud provider chosen (let $[x]$ denote the standard rounding function).

$$time_{execution,new} = \left[time_{execution,old} \cdot \frac{CloudScore_{new}}{CloudScore_{old}} \right] \tag{7.2}$$

For example, if the current provider is Oracle and the new one is Google, assuming an execution time of 1000 seconds on a vm type 2, it is possible to compute the desired value as

$$time_{execution,new} = \left[1000 \cdot \frac{7221}{4974} \right] = 1452s \tag{7.3}$$

Now, the results obtained will be presented by software, but there is a disclaimer: benchmarks have been run for all software, but not for all configurations possible. For this reason, some approximations have been made to continue with the thesis work and extract information from data, even if the results are incomplete. The ideal scenario is to collect data for all types of configurations on all cloud providers, to better shape the trend curves.

The graphs reported have two types of lines: solid and dotted. The solid curves represent the interval of interest used for the predictors, while the dotted ones are the projections of data outside that interval, just to show the trend for more or fewer cores: the trend lines are valid if the relationship doesn't change, so they were not considered in the predictors but shown here to give an idea of the scalability on a higher and lower number of cores.

Software 1

The benchmarks on Software 1 were run with two configurations: 64 and 192 cores. Starting from these two benchmarks and assuming the relationship between cores and time as linear, it is possible to draw the curves that represent the scalability of cloud providers; the linear dependence has been chosen because there was no other information, and so this was the only possible way to use those data.

As shown in Fig.7.1, all providers but Google increase their performance with the number of cores: this trend makes Google the choice to be avoided in case of more than one instance.

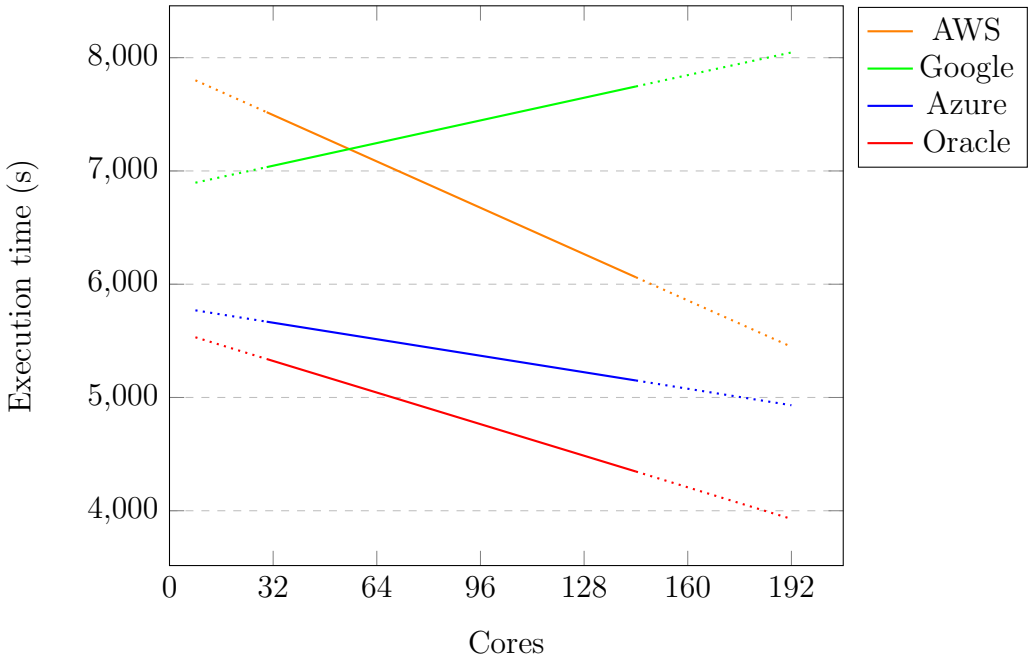

Figure 7.1: Software 1 instances' scaling

Software 2

The benchmarks on Software 2 have been run with 96 cores configurations on all providers, while more tests were conducted only with Oracle and Azure. With respect to the previous case, here the results on Google are better with 90 cores and so it is possible to consider that, in this specific case with this specific software, the performance should follow the number of cores. To draw the curves for AWS and Google, it has been used the same trend line of Azure: they use the same curve

parameters, but obviously with different reference data that are available. The Oracle curve is better described because the test conducted were more complete so it is possible to describe its scalability with more precision.

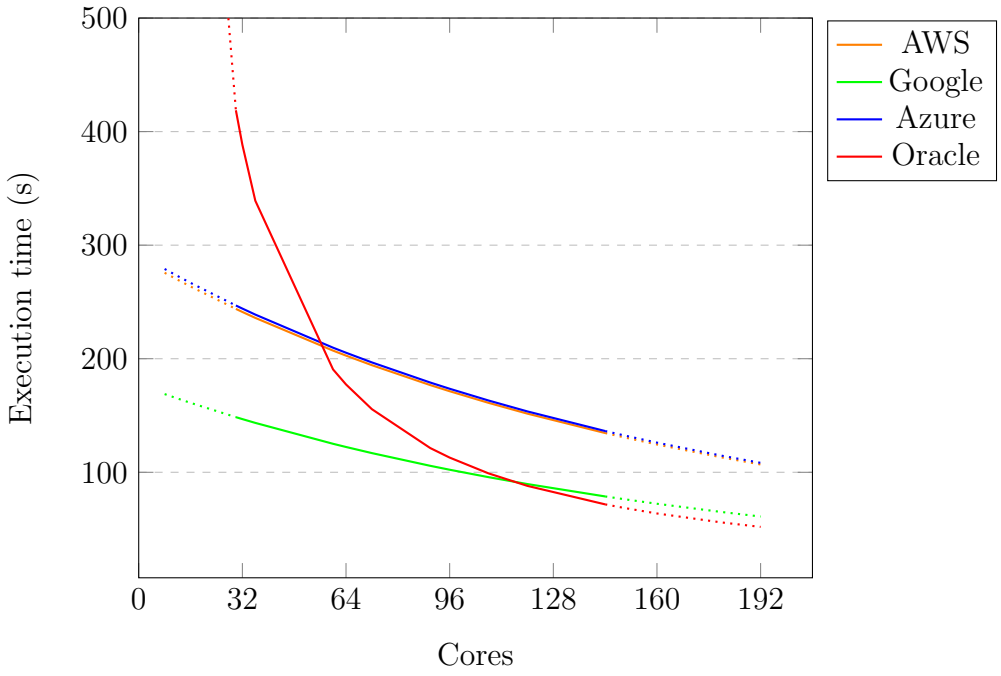

Figure 7.2: Software 2 instances' scaling

Software 3

The benchmarks on Software 3 have been run only with the 64 cores configuration, plus additional tests only on the Oracle platform. For this reason, AWS and Azure have been considered the same way as Oracle due to their ability to improve performance with the number of cores, modeling them with the same curve: this assumption has been done to extract a possible scalability model. For Google, the curve has been modeled differently, because it showed bad scalability and for this reason, the curve shows how the performance degrades with a higher number of cores: the parameter of this curve has been extracted from the Software 1 tests because no other clues were available to create a better model.

7.3 Machine Learning predictors

Both predictors have the same internal structure, composed of a set of machine learning algorithms, to exploit the differences between them and allow the choice of the best one, considering the one with the lowest Mean Absolute Percentage Error (MAPE). The predictor receives inputs, then based on what is the expected

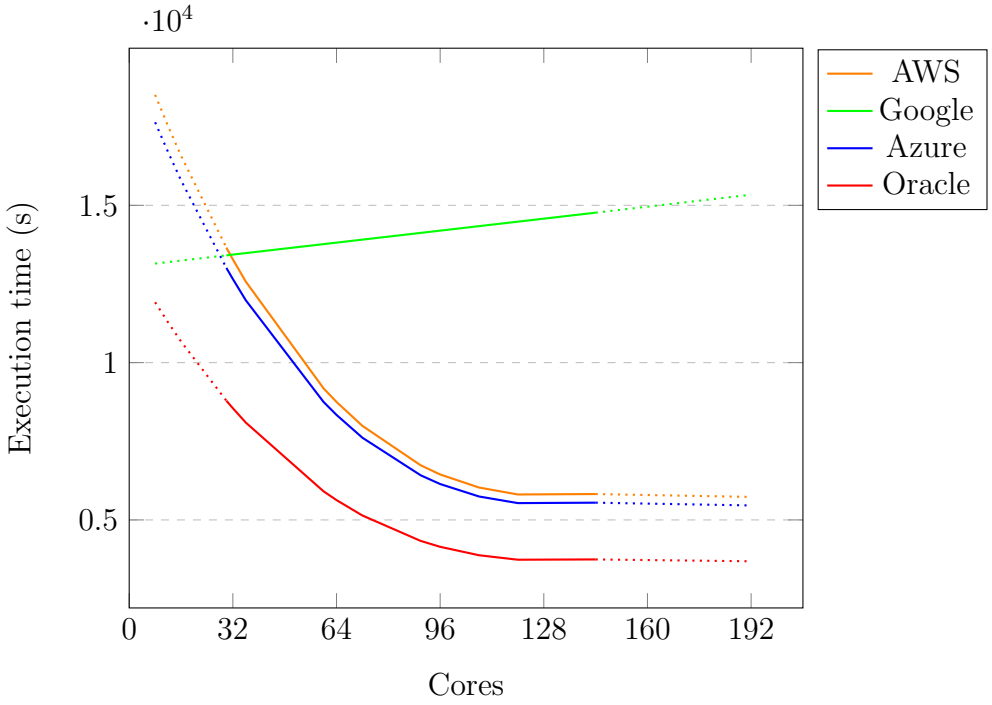

Figure 7.3: Software 3 instances' scaling

output, it retrieves from files the model marked as the best and uses it to predict the value, returning the output for the following part of the system.

7.3.1 Hyperparameter search

Machine learning algorithms have some parameters (called hyperparameters) that can be set to change how the algorithm learns and the training time required. A preliminary phase has been conducted to find a suitable range for each algorithm because it would be impossible to try all combinations. There are two types of search: random search and grid search. The random search extracts a random set of hyperparameters from the global set and uses it to train the model: it is possible to choose how many random extractions should be done, trying to cover the highest number of possible combinations, but obviously, at each run, the sets extracted will be different and so the training is not reproducible unless using always the same sets, but with this solution, the search will be fixed and no more random. The grid search is an exhaustive search in the whole space of the hyperparameters: it considers all the possible combinations of hyperparameters, resulting in a better model tuning, but with an increase in the search time proportional to the cardinality of sets and number of hyperparameter analyzed. The search is done for every machine learning algorithm, each one with its specific hyperparameters to be searched.

7.3.2 Training

When a set of hyperparameters is chosen, the model is trained using them and fed with the training set. The dataset is split into training and validation sets, using

a splitting function provided by the library used to implement the ML algorithms. The training phase proceeds until the maximum number of iterations has been reached or if the difference within the last 10 iterations is below a certain threshold (denoting that the algorithm is not improving anymore).

Each predictor is characterized by its inputs and its outputs, and they are taken from input tuples. The data passed to the predictors is the same, but different information is extracted: from them, the subset of inputs and the desired output are extracted and used for the training. Assuming the case of *cput* predictor, the inputs are all the pre-runtime parameters and the output is the *cput* itself, while the other information is discarded and not considered: the input set becomes $(cloud, software, cores, ram, vm\ type)$ and the output value is the *cput* value.

7.3.3 Validation

The validation set, obtained with the splitting function, is used to validate the model's training: this set is used to test if the model is learning and to provide its accuracy when the training is complete. This phase allows the model's validation, checking its performance results and using them for later comparison between different algorithms. A percentage of the dataset is used as the validation set and, in this case, the amount is 20%.

7.3.4 Performance evaluation

When a hyperparameter search iteration finishes, the error rate of that particular hyperparameter set is computed and stored; when all iterations are completed, there is a comparison between those values, and the best one determines which parameter set will be used for the final training. The performance evaluation used as a comparison value is the Mean Absolute Percentage Error (MAPE): this is an average error and for this reason, it suits the need of the system to choose the best ML algorithm.

7.4 Runtime parameters prediction

The runtime parameters predictor is not a single unit, but it is composed of 4 sub-predictors (one for each runtime parameter): each one is characterized by its inputs and its output. All predictors use the pre-runtime parameters as inputs, but they differ for the output: they can use internally a different algorithm to provide the result, and at the end, the data are aggregated in a tuple and fed forward to the system.

7.5 Runtime prediction

The runtime predictor uses as input the data from the previous predictors and the pre-runtime parameters, while its output is the runtime prediction. It differs from

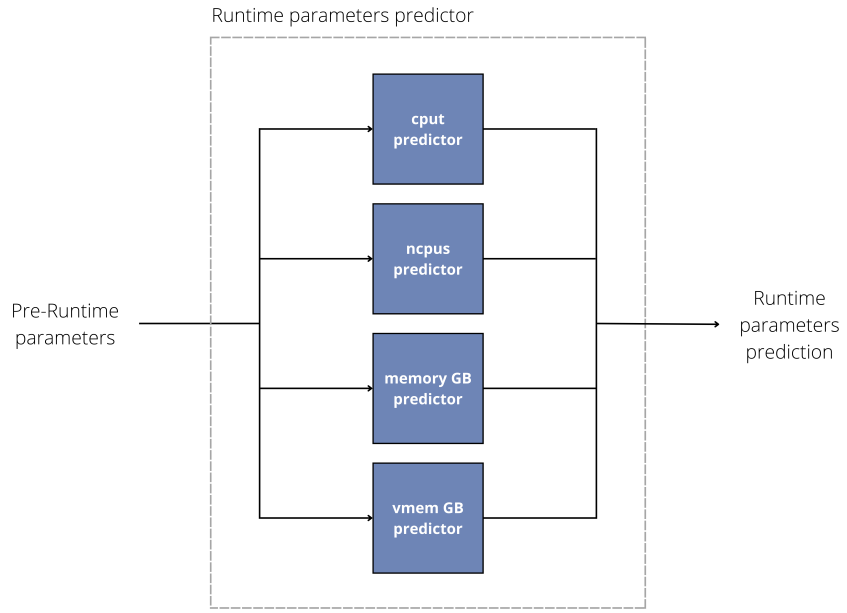

Figure 7.4: Runtime parameters predictors detailed

the previous prediction unit just for the input set and output, but the internal logic is the same.

7.6 CML (Continuous Machine Learning)

The training phase is done only when the system needs to learn from data, so when no model is ready to be used. Training models only once could be not effective when data starts to slightly change and so the results could become more imprecise with time. In order to correct the behavior of the model and to keep the performance, it is possible to exploit Continuous Machine Learning: the idea is to re-train models after n new run of simulations, to follow the real scenario of data and avoid missing information and provide wrong predictions. The training phase is done in the same way, and for this reason, after each re-training, the model used can change because the performance is evaluated again to choose the best one and not miss the chance to get the model that best fits the data.

This feature has been implemented to better follow the real data, considering also new entries that could potentially be different from the initial data distribution, and so the idea was to implement the so-called *learn on the job* feature inside the system that should learn while doing, potentially improving also performance: the improvement is not guaranteed, but the prediction error should stay around the initial value; the same error could increase if the feature is not implemented.

Chapter 8

Results

Two projects have been analyzed and used to test the system: *Project A* and *Project B* (names are anonymized to avoid data disclosure). A *project* describes a type of product and all related details: a product can be very different from the others, so data has been split to avoid interference, reducing the noise in predictions and thus getting more accurate results. Project A simulations have been run on the same day, while Project B ones have been run on more days: for this reason, the two representations will slightly differ to better illustrate the results and the considerations that it is necessary to do on those data.

The system can exploit 2 features introduced in chapter 7: **data augmentation** and **CML**. In order to get exhaustive results, all combinations of these 2 features have been tested, leading to 4 different scenarios, plus the reference one (labeled as *real*). Those scenarios are assigned with labels, in order to distinguish them but avoid reporting the description every time: A stands for augmentation, C for CML, and the N prefix means *not*.

- **Real**: the real cost of the simulations, computed using real runtime value from the dataset and multiplied by the instance cost; the system is not used here, this is the reference value.
- **NANC**: *No Augmentation, No CML*; no data augmentation and no CML (just one training at the beginning).
- **ANC**: *Augmentation, No CML*; data augmentation is present, but no CML.
- **NAC**: *No Augmentation, CML*; no data augmentation, but the CML is present (there is a re-training every n entries).
- **AC**: *Augmentation, CML*; data augmentation and CML are both present.

8.1 Comparison between ML algorithms performance

The performance has been evaluated for all 4 scenarios; it is possible to see from the graphs (reported below) that is not necessary that the runtime parameters

are very accurate because this is not the main point: the goal is to get the best runtime prediction, and the intermediate values are not displayed or used outside the system; the only value that could be interesting is the *ncpus* parameter because this could tell if the instance would be fully utilized or not, giving the opportunity to choose a smaller instance to reduce even more the final cost of the simulation. The algorithms used are reported using labels instead of the full names, and the mapping is reported in Table 8.1.

Acronym	ML algorithm
MLP	Multi-Layer Perceptron
SVR	Support-Vector Machine
LSVR	Linear Support-Vector Machine
RF	Random Forest
DT	Decision Tree
LR	Linear Regression
PR	Polynomial Regression
KN	K-Nearest Neighbors
MV	Mean Value

Table 8.1: ML labels and algorithms mapping

8.1.1 Runtime parameters

The runtime parameters' performance can be evaluated by looking at each algorithm for each scenario, considering the MAPE as the reference: the values reported have been rounded to avoid numbers with too many digits. The graph reported in Fig.8.1 shows the comparison between the *cput* parameter MAPE, leading to a very interesting result: using the data augmentation on the runtime parameters is not so useful and it leads to a bigger error, so it is not good if the system prediction would stop here. The reason behind the choice of *cput* parameter to show these results, it is because that one is directly related to the runtime value for the next predictor, so it could be an indicator for the global system, more than the other parameters, which are necessary for the system but don't show the same relationship with the runtime. In the case of the *ncpus* parameter (Fig.8.3), the values reported are 0 because the error is negligible, so the approximation led to 0 values.

8.1.2 Runtime output values

The runtime prediction uses again the MAPE as a reference for performance evaluation. The results consider the whole predictors' chain: this is the error of the whole system about its output. As it is possible to see, for some algorithms the data augmentation can help in reducing the average error, while in other cases it leads to a higher error.

Considering the relationship between *cput* and *runtime*, it is possible to see in Fig.8.5 that all parameters contribute to the final prediction, and so the average error is not dependent on a specific parameter but on the whole input set.

8.2 Predictions with CML

The system performs quite well on data using only one single training phase at the beginning, but the system can benefit from the CML and it is possible to notice this from both projects, A and B. The effect of the introduction of the CML can be seen not on the MAPE value, but in how the system performs on successive simulations, so in a time interval in which several runs have been scheduled and executed. Assuming this, there are two behaviors that the results show: Project A total cost can be reduced using the system, while Project B is an example of how the system can diverge from real data trends if it is not re-trained after n simulations, leading to a prediction that seems correct, but that becomes a huge cost increment while considering the real runtime of simulations.

The results are collected by using a set of simulations not seen by the system before, and kept separate for this test: the test set must not be shared with the system like the validation one to avoid fitting the system on a particular set of data; this approach allows understanding if the system is able to generalize when newer data is provided to it. There is a loop on the test set, such that for each iteration a simulation is passed to the system to predict the corresponding runtime. When the prediction is ready, this information is used to compute the predicted cost of the simulation: the system also identifies the best cloud provider to be used and if the predicted provider is the same used by the real simulation, the real runtime is used to compute the estimated cost, otherwise, if the providers differ the curves presented in the chapter 7 are exploited to estimate the running time of the simulation on the predicted provider, obtaining in such a way the estimated cost if the simulation would have run on the provider chosen by the prediction system. In the end, it is possible to compare the real cost of the simulations and compare it with the estimated cost, having a better way for the comparison of the 4 scenarios.

8.2.1 Project A

Project A simulations have been run on two consecutive days, so the results are reported only as the cumulative cost for those days: the reason is that a curve is not suitable because the granularity of the data is the day and so a bar representation is better. The graph in Fig.8.11 reports the comparison between the real total cost and the prediction using the system for all 4 scenarios: it is possible to notice how the cumulative predictions are close to the real value, with a $\Delta \sim 1\%$.

Exploiting the scalability curves of chapter 7, in Fig.8.12 is possible to see that the combination of data augmentation and CML led to a very interesting result: it is possible to optimize the total costs using the system. If the simulations would have scheduled with the prediction system, the total cost would have been reduced by 9.46%. The main advantage shown here is that the system could effectively reduce the total cost and help the scheduling of simulations on the right cloud provider to run them at the lowest cost.

Project A final results can lead to another result: the system MAPE could be lower without the data augmentation, but then looking at the cumulative cost, allows the system to better generalize over other data and predict a better cloud

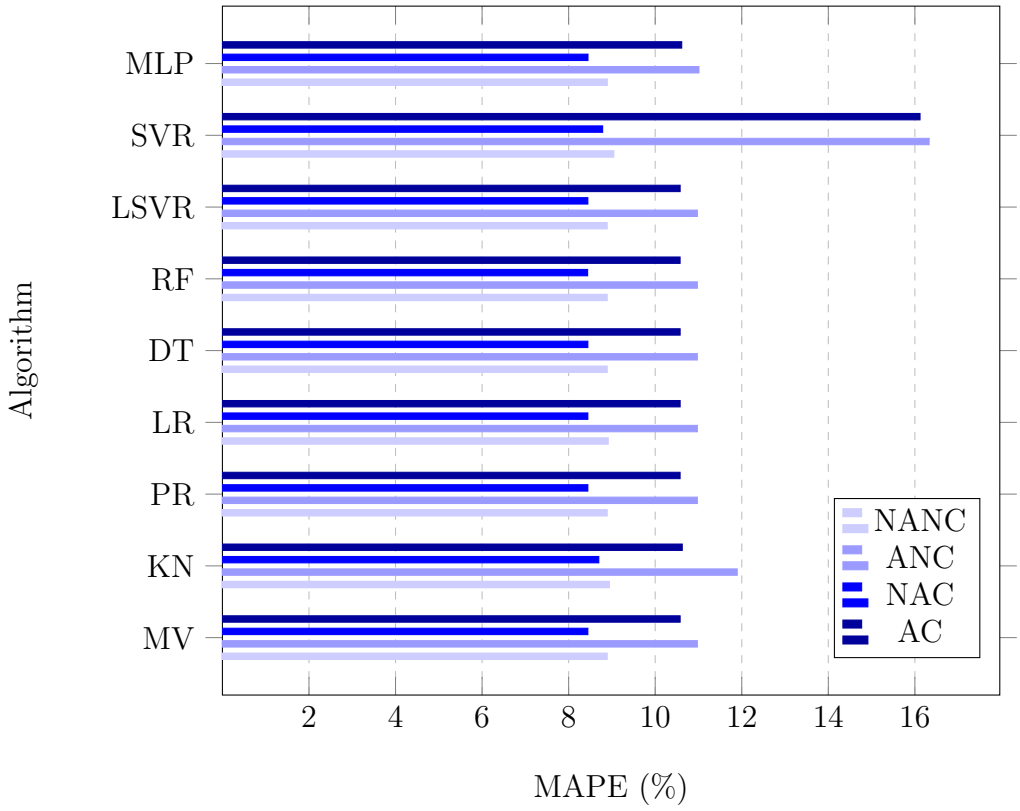

Figure 8.1: Project A cput

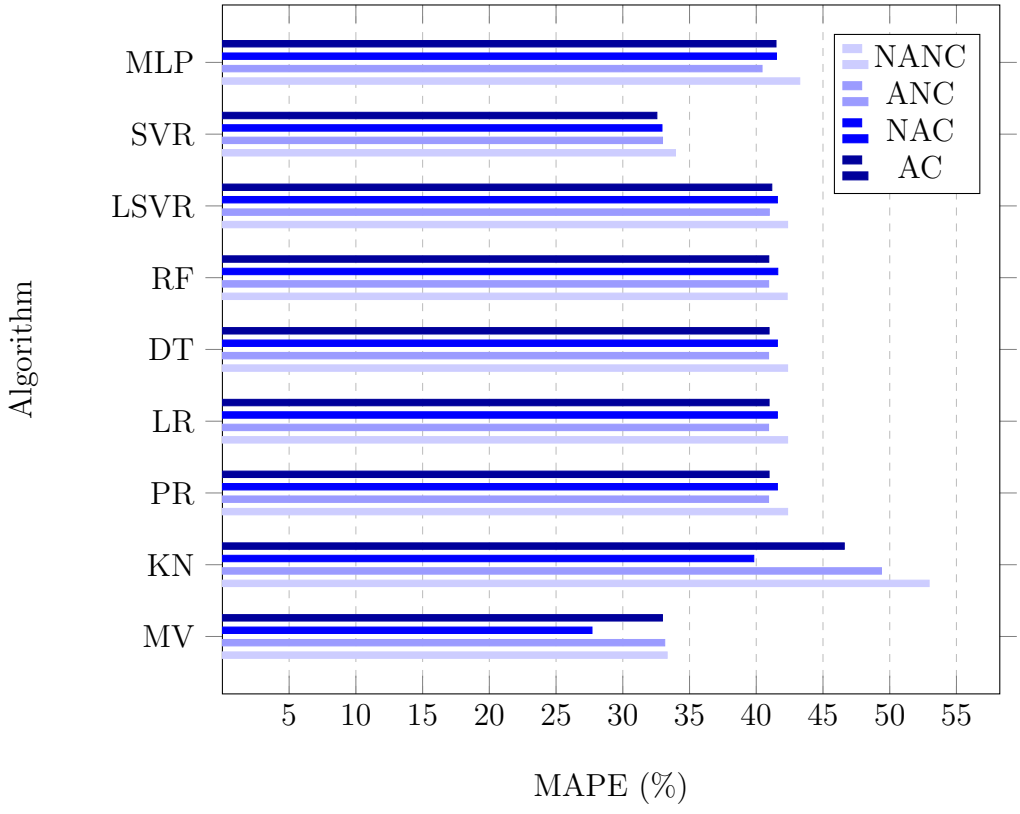

Figure 8.2: Project A memory GB

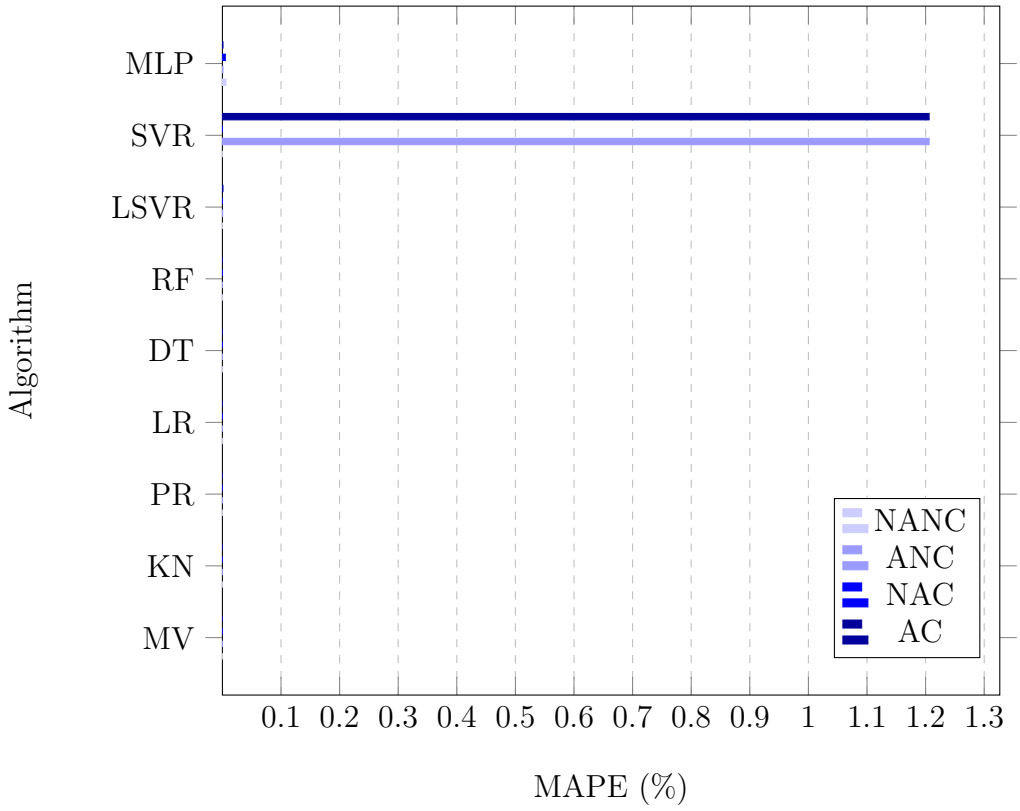

Figure 8.3: Project A ncpus

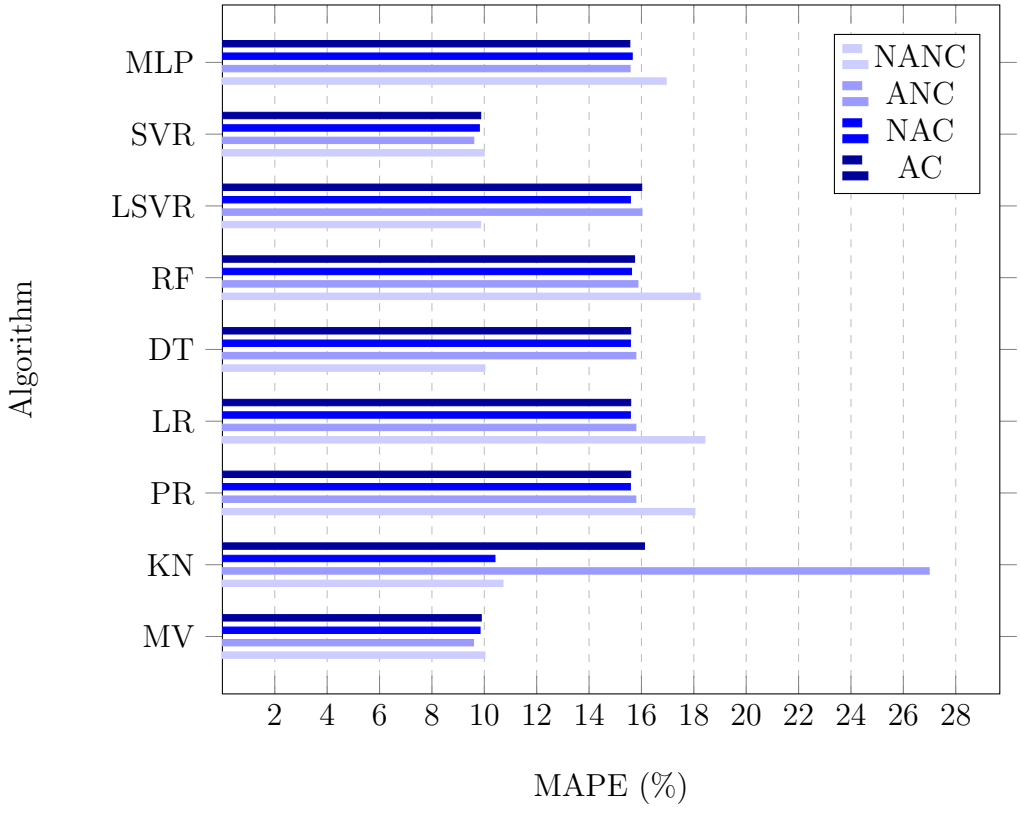

Figure 8.4: Project A vmem GB

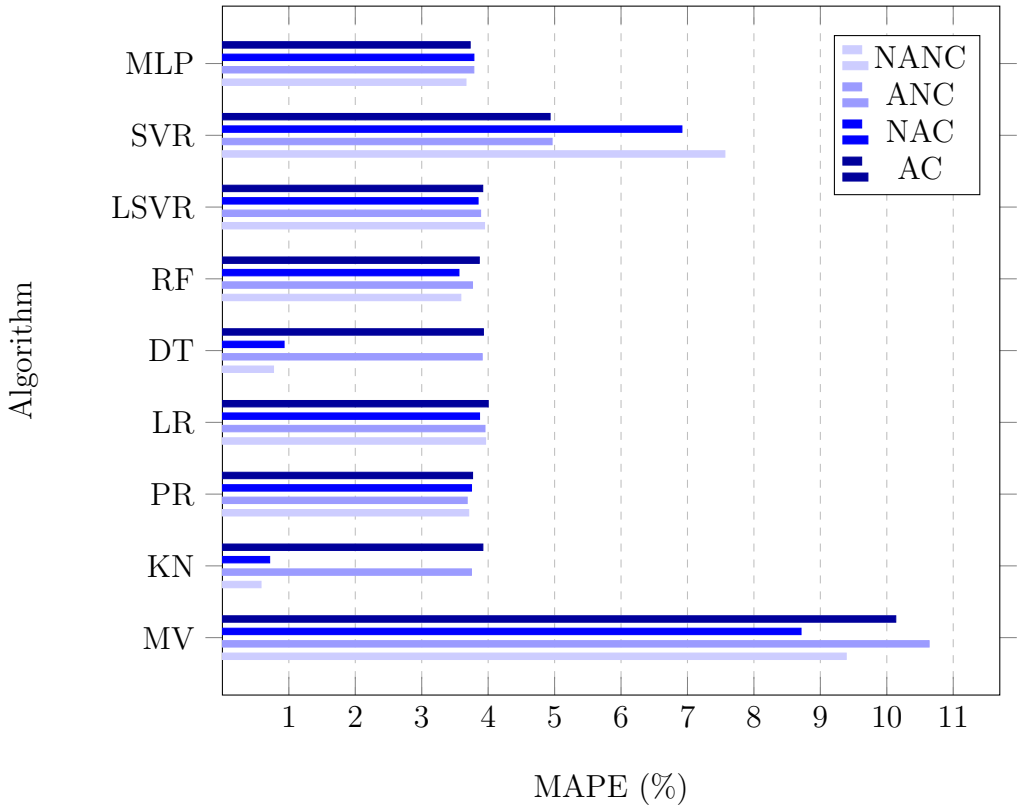

Figure 8.5: Project A runtime

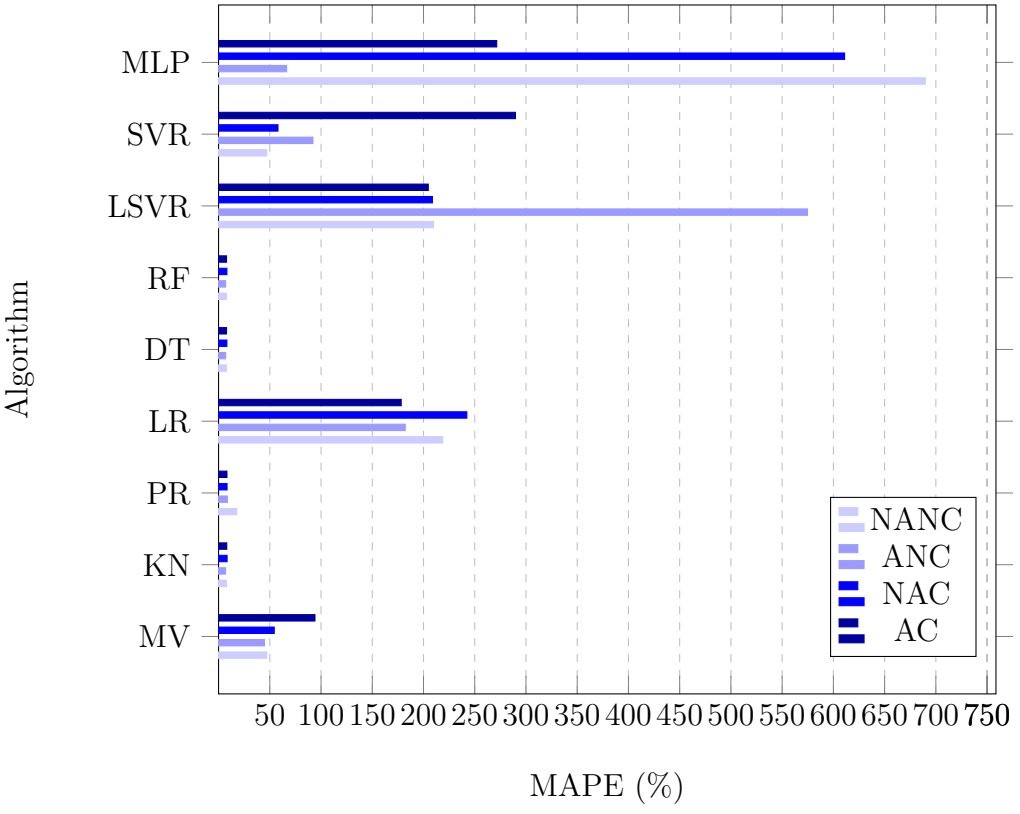

Figure 8.6: Project B cput

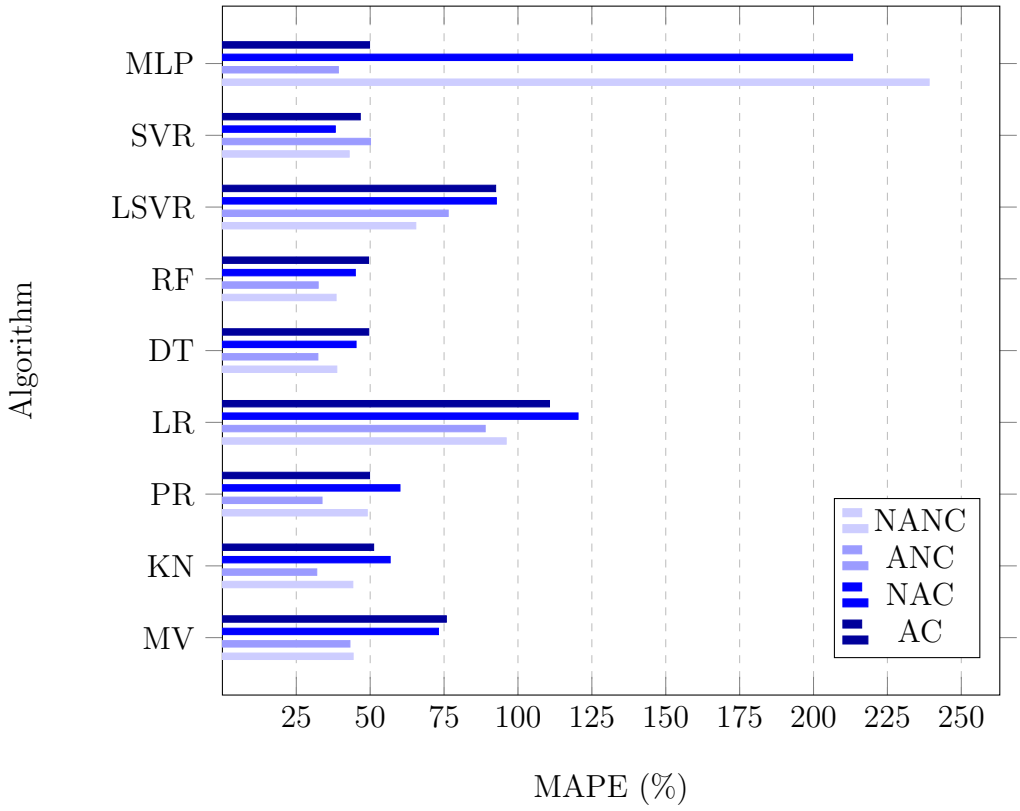

Figure 8.7: Project B memory GB

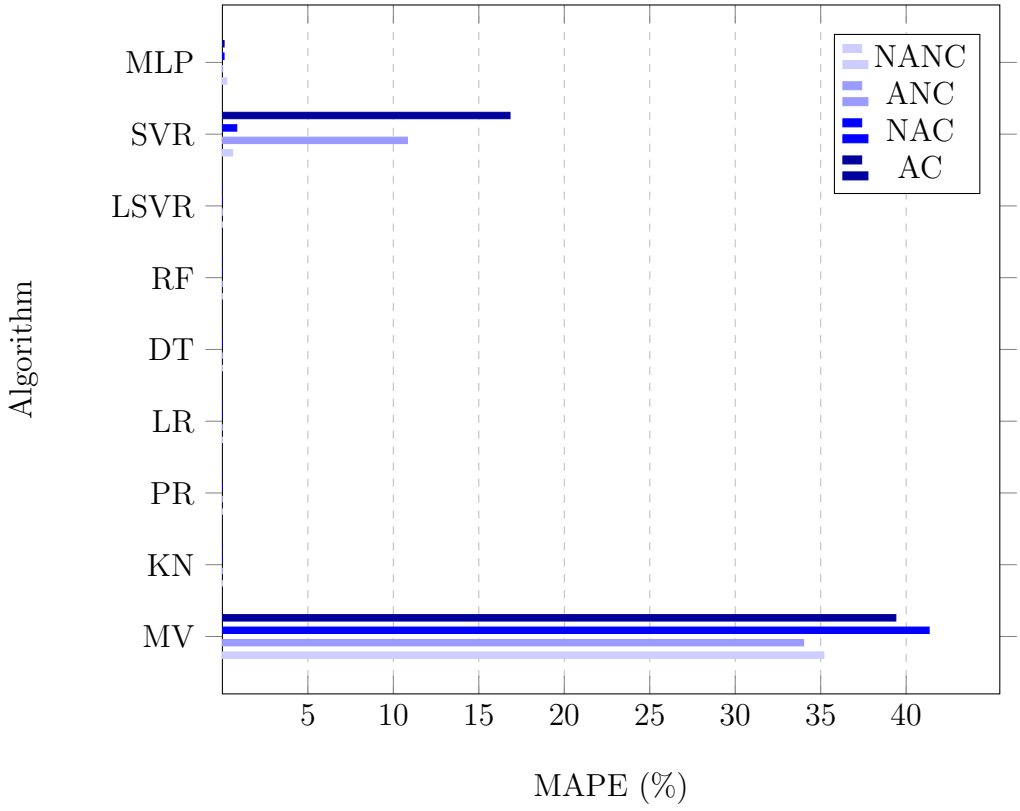

Figure 8.8: Project B ncpus

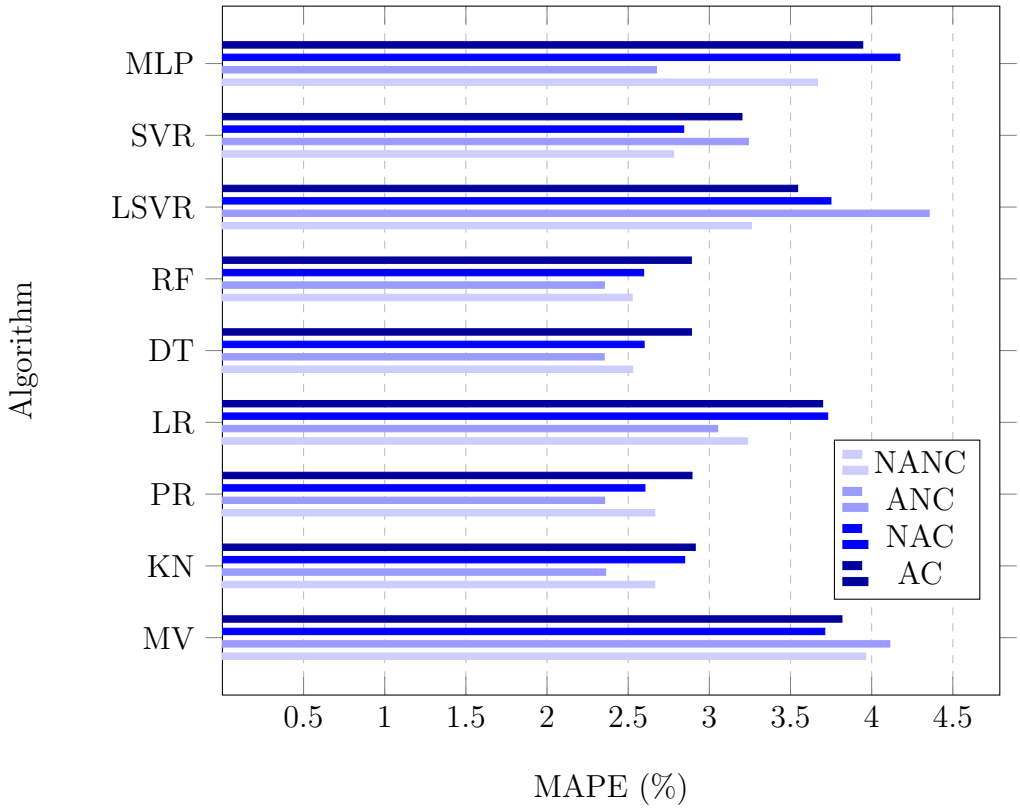

Figure 8.9: Project B vmem GB

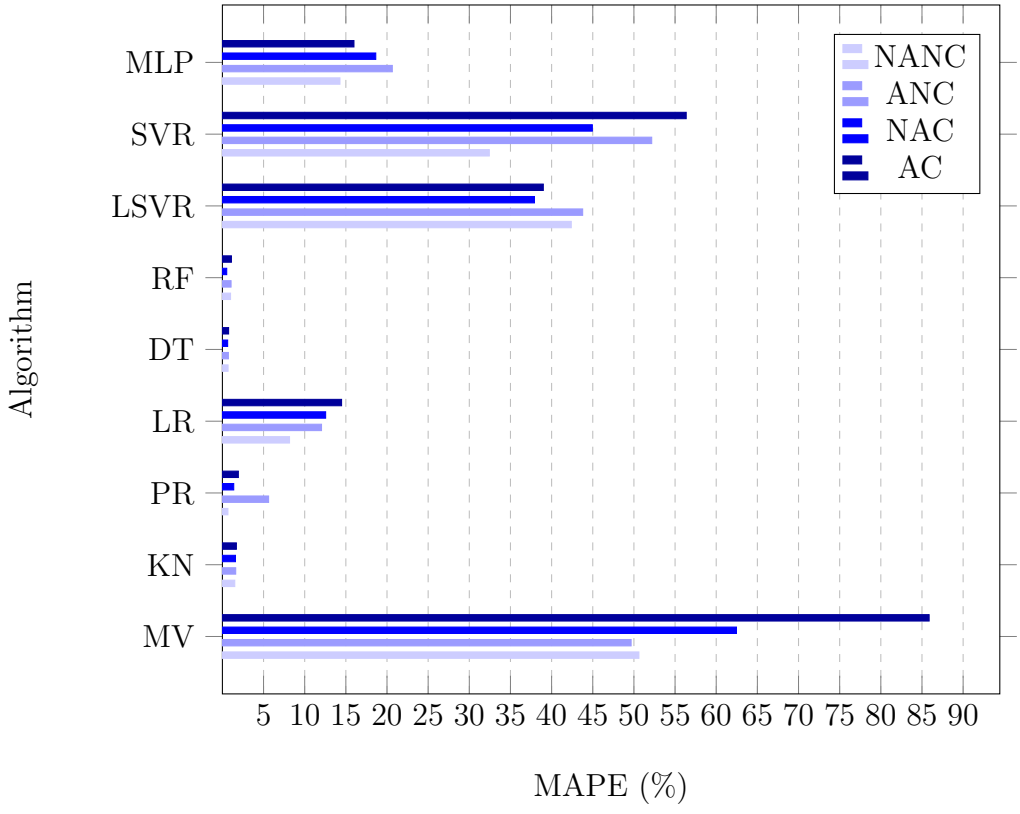

Figure 8.10: Project B runtime

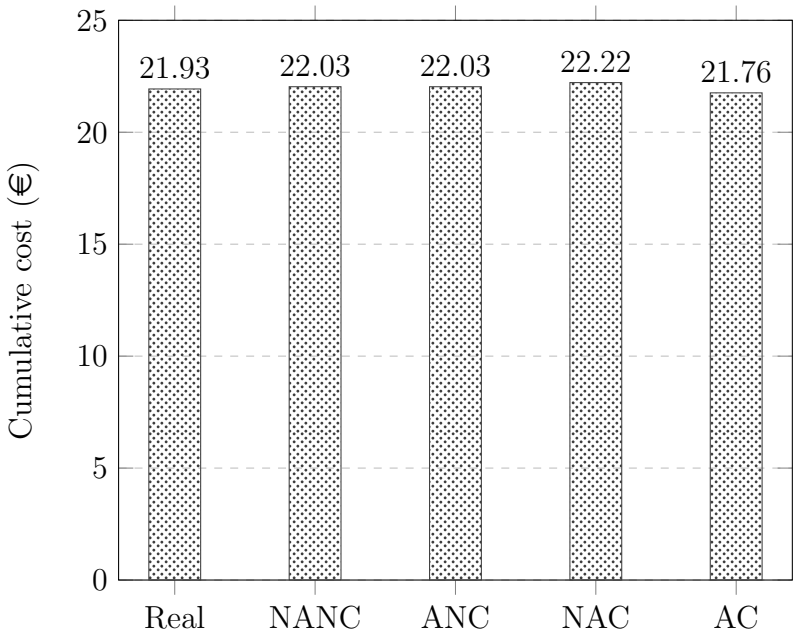

Figure 8.11: Cumulative costs, real vs. predicted

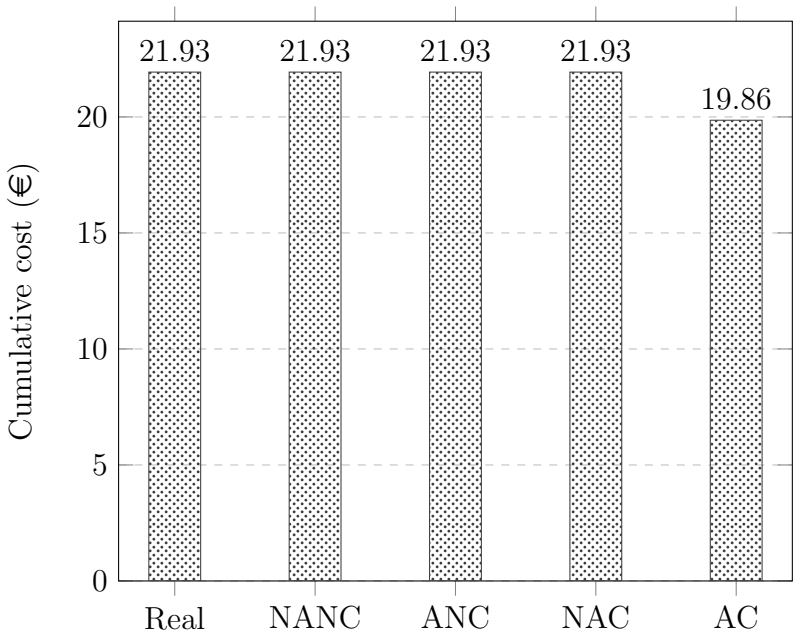

Figure 8.12: Cumulative costs, real vs. estimated

provider, which is not possible otherwise. The data augmentation increases the error, it is true, but the final achievement is better with it. This shows how machine learning algorithms should not overfit data and going below a certain error threshold the system would not be able to predict correct values for unseen examples (with respect to the seen ones during the training phase).

8.2.2 Project B

Project B will follow a different representation approach: the time interval considered is bigger than the other project, so a curve graph can better show the growth of costs during the time. As it is possible to see in Fig.8.13 the predictions of all 4 scenarios are quite similar, but these are *predictions* and not estimated values using the scalability curves of chapter 7: this disclaimer is needed because only by looking at these data the reader could be fooled thinking that the system will provide always correct results. The detail that changes how these results could be read is the chosen cloud provider because it is true that the predictions seem accurate, but the reality is different looking at estimated data.

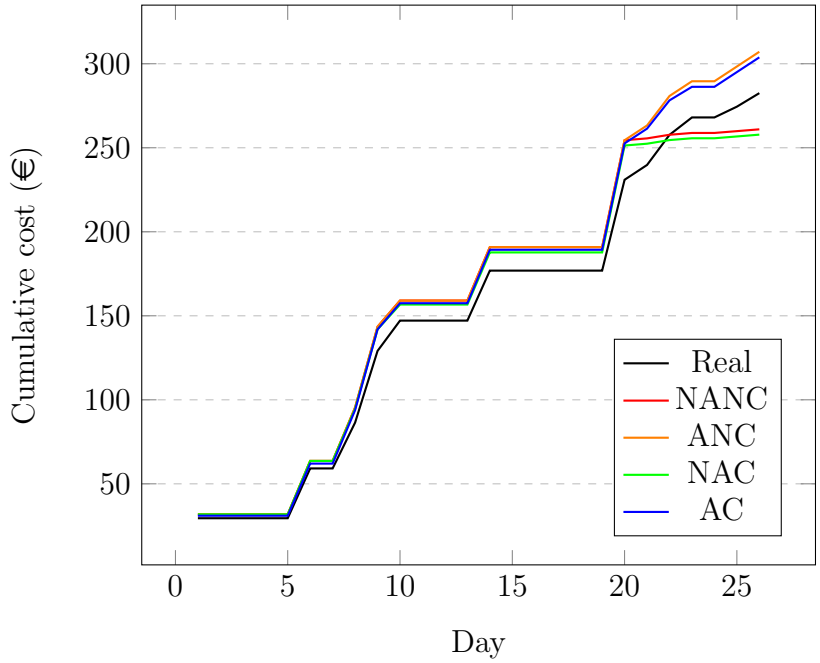

Figure 8.13: Cumulative costs, real vs. predicted

Project B is a very good example to show how the system seems working well, but when data starts to change it is no longer able to generalize and follow the new trend: at day 20 the simulations started to be run using the *vm type* parameter set to 3 instead of 4. The solution is the already known CML and the data augmentation, which helps in continuously considering new data when available and scaling on different providers, following the simulations during the time. In Fig.8.14 the comparison is between the real cumulative cost with the estimated costs: this is different from the comparison with predictions. In this case, the system is evaluated by looking at what effects it has on the scheduling and final cost and this is the main point: how the system is valuable when compared to the real case by using the estimations and not the predictions. The curves *NANC* and *NAC* overlap and they are the ones that from day 20 start diverging from the real curve. The curves *Real*, *ANC*, and *AC* overlap again, but they have the same trend.

In order to better present the results, a cumulative bar graph is reported in Fig.8.15: here the differences between scenarios are clearly visible.

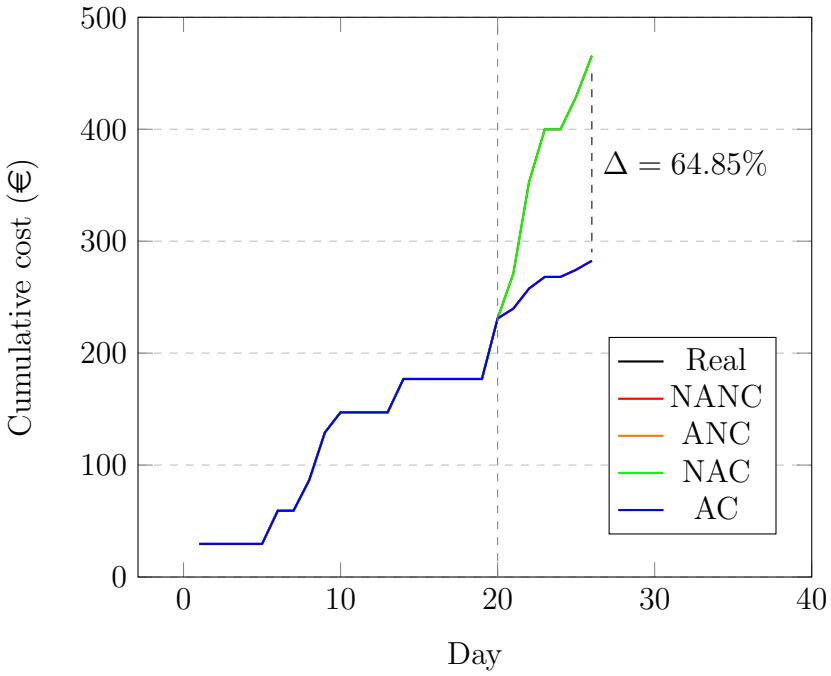

Figure 8.14: Cumulative costs, real vs. estimated

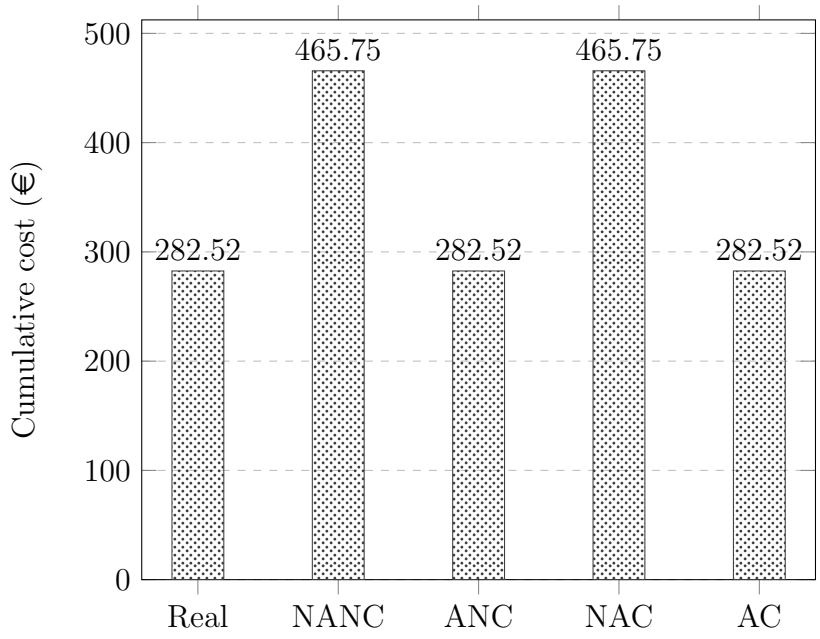

Figure 8.15: Cumulative costs, real vs. estimated

8.3 Summary

In conclusion, the results presented show two behaviors, one represented by Project A and the other by Project B. Project A showed that the system can provide a possible saving in terms of total costs using different cloud providers with respect to the one really used during the simulations' run: exploiting the prediction system, the potential saving achievable in this situation is about 9.46%, which is an interesting percentage. Project B, instead, showed how the system could provide

the wrong prediction if not re-trained and the data is augmented, resulting in a potentially higher cost up to 64.85% like in this case. Both projects have been chosen to show the benefits of the system and possible drawbacks: the prediction system should be kept updated and data should be augmented to get the best out of it.

Chapter 9

Conclusions

The system designed and developed to write this thesis can lead to a few key points about such implementation in a real-world scenario. Machine learning systems must be kept updated, learning from the new data and not only using information learnt at the beginning. Data used as input should be augmented not only if the number of training examples is not enough to train an ML system, but also to provide the latter with different examples crafted from the real ones: think about a project in which only one cloud provider is used to run simulations, so the system will be able to predict accurately only runtimes on that platform and have no clues for the others; this could fool the system because it has to improvise predictions when they use a different provider and obviously this will lead to an error that could become very big like in Project B.

Considering the data augmentation and CML benefits, ML systems are able to outperform human decisions when a lot of information that can be collected in tables is present: machines achieve much higher performance with this type of data called *structured*. The problems that could arise are evident, but the benefits are much more interesting than some drawbacks that could be mitigated by simply augmenting data and re-training the system when n new data entries are available.

9.1 Improvements

There are two main improvements that can be made to make the system more suitable for real-world scenarios and increase its performance and resiliency.

The first improvement that can be made is related to data collection from the scheduling system: the prediction unit has been implemented and designed based on available data, but the data collection was not designed to get data that would have been used to train machine learning algorithms. For this reason, the information collected could be changed to allow the prediction system to have more features in input, hoping for a performance improvement: more features do not mean adding tons of new information, but adding the right one that will characterize each simulation type, something that can allow a better generalization over new data.

The other improvement that can be done is running more benchmarks on cloud providers, to get more information about the runtime on different configurations (different instances or different numbers of the same one). If more benchmarks had been available, the scalability curves would have been more accurate, considering also the case at the end of the interval of interest and allowing the creation of a better model for cloud providers' estimation during data augmentation.

There is one more thing that could be done in a real working environment: use the system as a part of something bigger. The idea is not to stop on a single prediction system, but to use it to predict the runtimes of a project's simulations, have an idea of how much all of them will need to run, and proceed with a schedule in order to be more productive, like running on weekends in order to have results ready for the next week and start working on that.

Appendix A

Used tools

A.1 Sklearn library (Python)

Scikit-learn is an open-source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities [26].

A.1.1 Fitting and predicting: estimator basics

Scikit-learn provides dozens of built-in machine learning algorithms and models, called estimators. Each estimator can be fitted to some data using its fit method [26].

The fit method usually accepts 2 inputs:

- The samples matrix (or design matrix) X. The size of X is typically `(n_samples, n_features)`, which means that samples are represented as rows and features are represented as columns [26].
- The target values y, which are real numbers for regression tasks, or integers for classification (or any other discrete set of values). For unsupervised learning tasks, y does not need to be specified. y is usually a 1d array where the i-th entry corresponds to the target of the i-th sample (row) of X [26].

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> clf = RandomForestClassifier(random_state=0)
>>> X = [[ 1,  2,  3], # 2 samples, 3 features
...      [11, 12, 13]]
>>> y = [0, 1] # classes of each sample
>>> clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

Figure A.1: Fit a *RandomForestClassifier* to some very basic data [26]

```
>>> clf.predict(X) # predict classes of the training data
array([0, 1])
>>> clf.predict([[4, 5, 6], [14, 15, 16]]) # predict classes of
   new data
array([0, 1])
```

Figure A.2: Example of prediction [26]

Both X and y are usually expected to be *numpy* arrays or equivalent array-like data types, though some estimators work with other formats such as sparse matrices [26].

Once the estimator is fitted, it can be used for predicting target values of new data. You do not need to re-train the estimator, as shown in Fig.A.2 [26].

A.1.2 Automatic parameter searches

All estimators have parameters (often called hyperparameters in the literature) that can be tuned. The generalization power of an estimator often critically depends on a few parameters. For example a *RandomForestRegressor* has a `n_estimators` parameter that determines the number of trees in the forest, and a `max_depth` parameter that determines the maximum depth of each tree. Quite often, it is not clear what the exact values of these parameters should be since they depend on the data at hand [26].

Scikit-learn provides tools to automatically find the best parameter combinations (via cross-validation). In the example in Fig.A.3, we randomly search over the parameter space of a random forest with a *RandomizedSearchCV* object. When the search is over, the *RandomizedSearchCV* behaves as a *RandomForestRegressor* that has been fitted with the best set of parameters [26].

```
>>> from sklearn.datasets import fetch_california_housing
>>> from sklearn.ensemble import RandomForestRegressor
>>> from sklearn.model_selection import RandomizedSearchCV
>>> from sklearn.model_selection import train_test_split
>>> from scipy.stats import randint
...
>>> X, y = fetch_california_housing(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
   random_state=0)
...
>>> # define the parameter space that will be searched over
>>> param_distributions = {'n_estimators': randint(1, 5),
... 'max_depth': randint(5, 10)}
...
>>> # now create a searchCV object and fit it to the data
>>> search =
   RandomizedSearchCV(estimator=RandomForestRegressor(random_state=0),
... n_iter=5,
... param_distributions=param_distributions,
... random_state=0)
>>> search.fit(X_train, y_train)
RandomizedSearchCV(estimator=RandomForestRegressor(random_state=0),
   n_iter=5,
                   param_distributions={'max_depth': ...,
                                        'n_estimators': ...},
                   random_state=0)
>>> search.best_params_
{'max_depth': 9, 'n_estimators': 4}

>>> # the search object now acts like a normal random forest
   estimator
>>> # with max_depth=9 and n_estimators=4
>>> search.score(X_test, y_test)
0.73...
```

Figure A.3: Example of random search of hyperparameters [26]

Bibliography

[1] IBM, "What is cloud computing?." https://www.ibm.com/cloud/learn/cloud-computing

[2] F. Cicchiello, "Analysis, modeling and implementation of cost models for a multi-cloud Kubernetes context", December 2021

[3] AWS, "AWS VM shapes." https://aws.amazon.com/it/ec2/instance-types/

[4] Oracle, "Oracle VM shapes." https://docs.oracle.com/en-us/iaas/Content/Compute/References/computeshapes.htm

[5] Wikipedia, The Free Encyclopedia, "Regression analysis." https://en.wikipedia.org/wiki/Regression_analysis

[6] Wikipedia, The Free Encyclopedia, "Supervised learning." https://en.wikipedia.org/wiki/Supervised_learning

[7] Wikipedia, The Free Encyclopedia, "Multilayer perceptron." https://en.wikipedia.org/wiki/Multilayer_perceptron

[8] Wikipedia, The Free Encyclopedia, "Support-vector machine." https://en.wikipedia.org/wiki/Support-vector_machine

[9] Wikipedia, The Free Encyclopedia, "Random forest." https://en.wikipedia.org/wiki/Random_forest

[10] Wikipedia, The Free Encyclopedia, "Decision tree learning." https://en.wikipedia.org/wiki/Decision_tree_learning

[11] Wikipedia, The Free Encyclopedia, "Linear regression." https://en.wikipedia.org/wiki/Linear_regression

[12] Wikipedia, The Free Encyclopedia, "Polynomial regression." https://en.wikipedia.org/wiki/Polynomial_regression

[13] Wikipedia, The Free Encyclopedia, "k-nearest neighbors algorithm." https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[14] W. Zhang, "Machine learning approaches to predicting company bankruptcy", Journal of Financial Risk Management, vol. 06, 01 2017, pp. 364–374, DOI 10.4236/jfrm.2017.64026

[15] Wikipedia, The Free Encyclopedia, "Mode (statistics)." https://en.wikipedia.org/wiki/Mode_(statistics)

[16] Wikipedia, The Free Encyclopedia, "Median." https://en.wikipedia.org/wiki/Median

[17] PUNCH Torino S.p.A., "PUNCH Torino S.p.A. website." https://www.punchtorino.com/

[18] Do IT Systems S.r.l., "Do IT Systems S.r.l. website." https://www.doit-systems.it/

[19] Altair Engineering Inc., "Altair PBS Professional." https://www.altair.com/pbs-professional/

[20] Altair, "Altair pbs professional 2021.1.3 cloud guide." https://help.altair.com/2021.1.3/PBS%20Professional/PBSCloudGuide2021.1.3.pdf

[21] Microsoft, "Azure VM shapes." https://docs.microsoft.com/en-us/azure/virtual-machines/hbv3-series

[22] Google, "Google VM shapes." https://cloud.google.com/compute/vm-instance-pricing

[23] Wikipedia, The Free Encyclopedia, "Infiniband." https://en.wikipedia.org/wiki/InfiniBand

[24] Wikipedia, The Free Encyclopedia, "Remote direct memory access." https://en.wikipedia.org/wiki/Remote_direct_memory_access

[25] 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, 2020, pp. 256–268, DOI 10.1109/TCC.2017.2732344

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

[27] P. A. Abhang, B. W. Gawali, and S. C. Mehrotra, "Chapter 5 - emotion recognition", Introduction to EEG- and Speech-Based Emotion Recognition (P. A. Abhang, B. W. Gawali, and S. C. Mehrotra, eds.), pp. 97–112, Academic Press, 2016, DOI https://doi.org/10.1016/B978-0-12-804490-2.00005-1