Optimizing Quality-Aware Big Data Applications in the Cloud

Eugenio Gianniti, Michele Ciavotta, and Danilo Ardagna

Abstract—The last years witnessed a steep rise in data generation worldwide and, consequently, the widespread adoption of software solutions able to support data-intensive applications. Competitiveness and innovation have strongly benefited from these new platforms and methodologies, and there is a great deal of interest around the new possibilities that Big Data analytics promise to make reality. Many companies currently engage in data-intensive processes as part of their core businesses; however, fully embracing the data-driven paradigm is still cumbersome, and establishing a production-ready, fine-tuned deployment is time-consuming, expensive, and resource-intensive. This situation calls for innovative models and techniques to streamline the process of deployment configuration for Big Data applications. In particular, the focus in this paper is on the rightsizing of Cloud deployed clusters, which represent a cost-effective alternative to installation on premises. This paper proposes a novel tool, integrated in a wider DevOps-inspired approach, implementing a parallel and distributed simulation-optimization technique that efficiently and effectively explores the space of alternative Cloud configurations, seeking the minimum cost deployment that satisfies quality of service constraints. The soundness of the proposed solution has been thoroughly validated in a vast experimental campaign encompassing different applications and Big Data platforms.

Index Terms—G.1.6.h Nonlinear programming, C.4 Performance of Systems, C.2.4 Distributed Systems.

1 Introduction

Many analysts point out that we are experiencing years in which technologies and methodologies that fall within the sphere of Big Data have swiftly pervaded and revolutionized many sectors of industry and economy, becoming one of the primary facilitators of competitiveness and innovation [1].

IDC reported that the Big Data market will grow from \$150.8 billion in 2017 to \$210 billion in 2020, with a compound annual growth rate of 11.9% [2]. Big Data applications offer many business opportunities that stretch across industries, especially to enhance performance, as in the case of data-driven decision support systems. Furthermore, data-intensive applications (DIAs) can also prove extremely useful in social intervention scenarios to improve the overall life quality of a community, as happens with systems for predicting, preventing, and developing policies in the event of natural disasters.

Undeniably, one of the pillars underpinning the Big Data revolution is the MapReduce paradigm, which has enabled massive-scale parallel analytics [3]. MapReduce is the core of Apache Hadoop, an open source framework that has proven capable of managing large datasets over either commodity clusters or high performance distributed topologies [4]. Hadoop attracted the attention of both academia and industry as it overtook the scalability limits of traditional data warehouse and business intelligence solutions [3]. For the first time, processing unprecedented amounts of structured and unstructured data was within reach, thus opening up a whole world of opportunities.

- *E. Gianniti and D. Ardagna are with Politecnico di Milano. Milan, Italy*
  Email: {name.lastname}@polimi.it
- *M. Ciavotta is with Università di Milano-Bicocca, Milan, Italy*
  Email: michele.ciavotta@unimib.it

Despite the fact that many new solutions have been created over time, Hadoop has been able to age well, constantly renewing itself to support new technologies (e.g., SSD, caching, I/O barriers elimination) and workloads (batch and interactive) [5]. In addition, a large Hadoop-based ecosystem of highly specialized tools has sprung up and matured. Consequently, for a long time it has been the foremost solution in the Big Data scene. This is confirmed by the fact that, only a few years ago, more than half of the world's data were somehow processed via Hadoop [6].

Paradoxically, the MapReduce paradigm, which has contributed so much to Hadoop's rise, is steadily declining in favor of solutions based on more generic and flexible processing models. Among these, Apache Spark is a framework that is enjoying considerable success in the area of batch processing, whilst Apache Kafka and Flink are valuable alternatives for data streaming applications [7].

In spite of the mentioned points in favor of Big Data technologies, fully embracing them is a very complex and multifaceted process. From the technological point of view, many efforts have been made to make it more accessible, but establishing a production-ready deployment is time-consuming, expensive, and resource-intensive. Not to mention the fact that fine-tuning is still often perceived as a kind of occult art.

It is widely held that there is a clear need for an *easy button* to accelerate the adoption of Big Data analytics [8]. This is why many companies have started offering Cloud-based Big Data solutions (like Microsoft HDInsight, Amazon Elastic MapReduce, or Google Cloud Dataproc) and IDC expects that nearly 40% of Big Data analyses will be supported by public Clouds by 2020 [9]. The advantages of this approach are manifold. For instance, it provides an effective and cheap solution for storing huge amounts of data, whereas the pay-per-use business model allows to cut upfront expenses and reduce cluster management costs. Moreover, the elasticity can be harnessed to tailor clusters capable to support DIAs in a cost-efficient fashion. Yet, provisioning workloads in a public Cloud environment entails several challenges. In particular, the space of configurations (e.g., node types and number) can be very large, thus identifying the exact cluster configuration is a demanding task [10], especially in the light of the consideration that the blend of job classes in a specific workload and their resource requirements may also be time-variant.

At the very beginning, MapReduce jobs were meant to run on dedicated clusters to support batch analyses via a FIFO scheduler [11], [12]. Nevertheless, DIAs have evolved and nowadays large queries, submitted by different users, need to be performed on shared clusters, possibly with some guarantees on their execution time [4], [13]. However, guaranteeing job performance implies being able to predict with a sufficient degree of accuracy the execution times in shared environments, and this is one of the greatest challenges in Big Data due to the large number of variables at stake [14], [15]. Therefore, in such systems, capacity allocation becomes one of the most important aspects.

Determining the optimal number of nodes in a cluster shared among multiple users performing heterogeneous tasks is a relevant and difficult problem [10], [15]. In particular, in this work the problem of rightsizing Apache Hadoop and Spark clusters, deployed in the Cloud and supporting different job classes while guaranteeing the quality of service (QoS) is addressed and solved. Each class is associated with a given concurrency level and a deadline, meaning that several similar jobs have to run on the cluster at the same time without exceeding a constraint imposed on their execution times. Additionally, the cluster can exploit the flexibility of Cloud infrastructures by adopting different types of virtual machines (VMs), taking into account performance characteristics and pricing policies.

The capacity planning problem is formulated by means of a mathematical model, with the aim of minimizing the cost of Cloud resources. The problem considers several VM types as candidates to support the execution of DIAs from multiple user classes. Cloud providers offer VMs of different capacity and cost. Given the complexity of virtualized systems and the multiple bottleneck switches that occur in executing DIAs, very often the largest available VM is not the best choice from either the performance or performance/cost ratio perspective [4], [10]. Through a search space exploration, the proposed approach seeks the optimal VM type and number of nodes considering also specific Cloud provider pricing models (namely, reserved, on demand, and spot instances [16]). The underlying optimization problem is NP-hard and is tackled by a simulation-optimization procedure able to determine an optimized configuration for a cluster managed by the YARN Capacity Scheduler [17]. DIA execution times are estimated by relying on multiple models, including machine learning (ML) and simulation based on queueing networks (QNs), stochastic Petri nets (SPNs) [18], as well as an ad hoc discrete event simulator, dagSim [19], especially designed for the analysis of applications involving a number of stages linked by directed acyclic graphs (DAGs) of precedence constraints. This property is common to legacy MapReduce jobs, workloads based on Apache Tez, and Spark-based applications.

Our work is one of the first contributions facing the design time problem of rightsizing data-intensive Cloud systems adopting the Capacity Scheduler. In particular, it builds upon our previous research presented in [20] and provides a thorough description of D-SPACE4Cloud[1], a software tool designed to help system administrators and operators in the capacity planning of shared Big Data clusters hosted in the Cloud to support both batch and interactive applications with deadline guarantees. With respect to previous releases, the tool now supports, besides classical MapReduce workloads, also Spark applications, which feature generic DAGs execution models. The generalization of the addressed problem has required significant changes in the approach implemented by the tool; in fact, D-SPACE4Cloud now exploits ML instead of scheduling-theory-based formulas and integrates dagSim[2], a purpose-built discrete event simulator capable of handling highly parallel applications with DAGs of precedence constraints. Moreover, the optimization core currently features a new search technique, more effective and less likely to return local optima.

Finally, the present paper also reports extensive validation results obtained on a gamut of real deployments. In particular, experiments based on the TPC-DS industry benchmark for business intelligence data warehouse applications are presented and the outcomes discussed. Microsoft Azure HDInsight, Amazon EC2, the CINECA Italian supercomputing center, and an in-house cluster based on IBM POWER8 processors have been considered as target deployments. The proposed approach proved to achieve good performance across all these alternatives, despite their peculiarities. Simulation results and experiments performed on real systems have shown that the achieved percentage error is within 30% of the measurements in the very worst case, with an average error around 12% for QNs and as low as 3% when using dagSim. On top of this, it is shown that optimizing the resource allocation, in terms of both type and number of VMs, offers savings up to 20–30% in comparison with the second best configuration. In particular, at times, general purpose instances turned out to be a better alternative than VMs advertised as suitable for Big Data workloads. Finally, with respect to an alternative method integrating other state of the art proposals facing the capacity planning problem [21], [22], we demonstrate to identify more accurate solutions, with an average percentage error of about 3% on the number of cores with respect to 7–16%.

This paper is organized as follows. Section 2 overviews D-SPACE4Cloud's architecture. Section 3 presents in detail the problem addressed in the paper. In Section 4 the focus is on the formulation of the optimization problem and on the design time exploration algorithm implemented within the tool. In Section 5 our approach is evaluated by considering first the accuracy that can be achieved by the simulation models and then the overall effectiveness of the optimization method. Finally, Section 6 compares this work with other proposals available in the literature and Section 7 draws the conclusions.

2 D-SPACE4Cloud Architecture

The tool we present and discuss in this paper, namely D-SPACE4Cloud, has been developed within the context of

1. D-SPACE4Cloud is available as open source under the Apache license version 2.0 at: https://github.com/dice-project/DICE-Optimisation-Back-End.

2. https://github.com/eubr-bigsea/dagSim

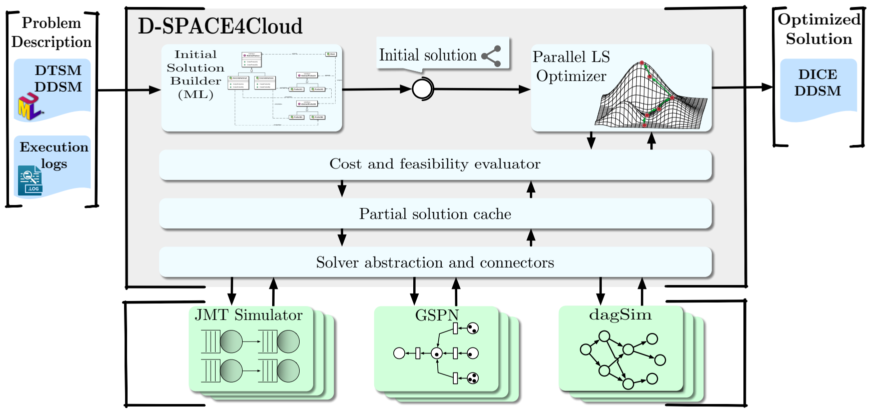


Figure 1. D-SPACE4Cloud architecture

the DICE H2020 EU research project [23]. The project aims at filling gaps in model-driven engineering with regard to the development of DIAs in Cloud environments, embracing the DevOps [24] culture, developing an integrated ecosystem of tools and methodologies intended to streamline the DIA development through an iterative and quality-aware approach (design, simulation, verification, optimization, deployment, and refinement). DICE primarily proposes a data-aware UML profile that provides designers with the means necessary to model dynamic and static characteristics of the data to be processed as well as the impact on the performance of application components. In addition, the project develops an IDE capable of supporting managers, developers, and operators in quality-related decisions. The IDE enforces the iterative design refinement approach through a toolchain of both design (simulation, verification, and optimization of deployment) and run time tools (deployment, testing, and feedback analysis of monitoring data).

D-SPACE4Cloud is the deployment optimization solution provided by the DICE IDE. The tool serves the purpose of optimizing the deployment costs for one or more DIAs with a priori performance guarantees. In a nutshell, within the quality-aware development process envisioned by DICE, a DIA is associated with QoS requirements expressed in form of a *maximum execution time* (deadline) and *concurrency level* (several users executing the same application at the same time with a certain think time). D-SPACE4Cloud addresses and solves the capacity planning problem consisting in the identification of a minimum cost cluster (both for public and private Clouds) supporting concurrent and on-time execution of DIAs. To this end, the tool implements a design time exploration algorithm able to consider multiple target VM candidates also across different Cloud providers.

Figure 1 depicts the main elements of the D-SPACE4Cloud architecture. Our tool is a distributed software system designed to exploit multi-core and multi-host architectures to work at a high degree of parallelism. In particular, it features a presentation layer (integrated in the IDE) devoted to manage the interactions with users and with other components of the DICE ecosystem, an optimization service (colored gray), which transforms the inputs into suitable performance models [18] and implements the optimization strategy, and a horizontally scalable assessment service (colored green in the picture), which abstracts the performance evaluation from the particular solver used. Currently, a QN simulator (JMT [25]), a SPN simulator (GreatSPN [26]), and a discrete event simulator (dagSim [19]) are supported.

D-SPACE4Cloud takes as input the following data:

1) a UML description of the applications provided via *DICE Platform and Technology Specific Models* (DTSMs) [24] or execution logs obtained executing the applications in a pre-production environment.
2) a partially specified deployment model for each application. The deployment model must be specified in *DICE Platform, Technology, and Deployment Specific Model* (DDSM) format.
3) a description of the execution environment, that is a list of candidate providers and VM types along with VM performance profiles.
4) the list of QoS constraints, that is the concurrency level and deadline for each DIA, respectively.

The optimization service is the tool centerpiece. It primarily parses the inputs, stores the relevant information using a more manageable and compact format, then calculates an initial solution for the problem (via the *Initial Solution Builder*) and improves it via a simulation-optimization algorithm (implemented by the *Parallel Local Search Optimizer*).

The initial solution is generated by solving a mixed integer nonlinear programming (MINLP) formulation where a subset of constraints have been modeled by training suitable ML models. More details are available in Section 4.1. Nevertheless, the initial solution can still be improved, mainly because the MINLP harnesses an approximate representation of the application-cluster liaison. More accurate performance models (e.g., QNs and SPNs) are therefore used to create room for further cost reduction; however, since the flip side of accuracy is a time-consuming evaluation process, the solution space has to be explored in the most efficient way, especially avoiding the evaluation of unpromising configurations. The Optimizer component carries out this task, implementing a simulation-optimization technique applied independently, and in parallel, on all the application classes.

3 Problem Statement

In this section the aim is to introduce some important details on the problem addressed in this work. The envisioned scenario consists in setting up a cluster to efficiently carry out a set of interactive DIAs. A cluster featuring the YARN Capacity Scheduler and running on a public infrastructure as a service (IaaS) Cloud is considered as target deployment. In particular, the cluster has to support the simultaneous execution of DIAs in the form of Hadoop jobs and Hive, Pig, or SparkSQL queries. Different classes $\mathcal{C} = \{i \,|\, i = 1, \ldots, n\}$ gather the applications that exhibit a similar behavior and performance requirements. The cluster composition and size, in terms of type and number of VMs, must be decided in such a way that, for every class i, h_i jobs are guaranteed to execute concurrently and complete before a prearranged deadline D_i. Moreover, YARN is configured in a way that all available cores can be dynamically assigned for task execution. Finally, in order to limit the risk of data corruption, and according to the practices suggested by major Cloud vendors, the datasets reside on a Cloud storage service accessible in quasi-constant time.

In general, IaaS providers feature a large catalog of VM configurations that differ in features (CPU frequency, number of cores, available memory, etc.) and prices. Making the right design decision implies a remarkable endeavor, yet it can be rewarded by important savings throughout the cluster life cycle. Let us index with j the VM types available across, possibly, different Cloud providers and let $\mathcal{V} = \{j \,|\, j = 1, \ldots, m\}$. We denote by τ_i the VM type used to support DIAs of class i and with ν_i the number of VMs of that kind allocated to class i.

In this scenario, the pricing model is inspired by Amazon EC2's [16]. The provider offers: 1) *reserved* VMs, for which it adopts a one-time payment policy that grants access to a certain number of them at a discounted fare for the contract duration; 2) *on demand* VMs, which can be rented by the hour according to current needs; and 3) *spot* VMs, created out of the unused datacenter capacity. Such instances are characterized by discounted fees at the expense of reduced guarantees on their reliability. In order to obtain the most cost-effective configuration, the assumption is to exploit reserved VMs (denoting with r_i their number) to satisfy the bulk of computational needs and complement them with on demand (d_i) and spot (s_i) instances ($\nu_i = r_i + d_i + s_i$). Let ρ_{τ_i}, δ_{τ_i}, σ_{τ_i} be the unit price for VMs of type τ_i, respectively, reserved, on demand, and spot. Overall, the cluster hourly renting out costs can be calculated as follows:

$$\sum_{i \in \mathcal{C}} \left(\rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i \right). \tag{1}$$

As the reliability of spot VMs is susceptible to the Cloud provider's workload fluctuations, to keep a high QoS level the number of spot VMs is bounded not to be greater than a fraction η_i of the total number of VMs ν_i for each class i. In addition, reserved VMs must comply with the long term contract signed with the Cloud provider and cannot exceed the prearranged allotments R_{ij}, that is every class is associated with a separate pool of reserved VMs of any type. It is worth noting that this cost model is general enough to remain valid, zeroing the value of certain parameters, even when considering IaaS providers with pricing policies that do not include reserved or spot instances.

In the remainder, c_i denotes the total number of YARN containers devoted to application i, whilst m_i and v_i are the container capacities in terms of RAM and vCPUs, and M_j and V_j represent the total RAM and vCPUs available in a VM of type j.

The problem faced in this work is to reduce the operating costs of the cluster by efficiently using the virtual resources in lease. This translates into a resource provisioning problem where renting out costs must be minimized subject to the fulfillment of QoS requirements, namely a per-class concurrency level h_i given certain deadlines D_i. In the following we assume that the system supports h_i users for each class and that users work interactively with the system and run another job after a think time exponentially distributed with mean Z_i, i.e., the system is represented as a closed model [27].

In order to rigorously model and solve this problem, it is crucial to predict with fair confidence the execution times of each application class under different conditions: level of concurrency, size, and composition of the cluster. The execution time can generically be expressed as:

$$T_i = \mathcal{T}_i \left(\nu_i, h_i, Z_i, \tau_i \right), \quad \forall i \in \mathcal{C}. \tag{2}$$

What is worthwhile to note is that the previous formula represents a general relation describing either closed form results, as those presented in [15], based on ML [28], or the average execution times achieved via simulation: in this paper both the latter approaches are adopted.

Since the execution of jobs on a sub-optimal VM type may lead to performance disruptions, it is critical to avoid assigning tasks belonging to class i to the wrong VM type $j \neq \tau_i$. Indeed, YARN allows for specifying node labels and partitioning nodes in the cluster according to these labels, then it is possible to enforce such separation. The optimized configuration statically splits different VM types with this mechanism and adopts within each partition either a further static separation in classes or a work conserving scheduling mode, where idle resources can be assigned to jobs requiring the same VM type. The choice about the scheduling policy within partitions is not critical, since it does not impact on the optimization technique or performance prediction approach. When resources are tightly separated, the performance estimate is expected to accurately mirror the real system behavior, whilst in work conserving mode the observed performance may improve due to a better overall utilization of the deployed cluster, hence the prediction is better interpreted as a conservative upper bound. Equation (2) can be used to formulate the deadline constraints as:

$$T_i \leq D_i, \quad \forall i \in \mathcal{C}. \tag{3}$$

In light of the above, the ultimate goal of the proposed approach is to identify the optimal VM type selection τ_i, and type of lease and number of VMs ($\nu_i = r_i + d_i + s_i$) for each class i, such that the total operating costs are minimized while the deadlines and concurrency levels are met.

The reader is referred to Figure 2 for a graphical overview of the main elements of the considered resource provisioning

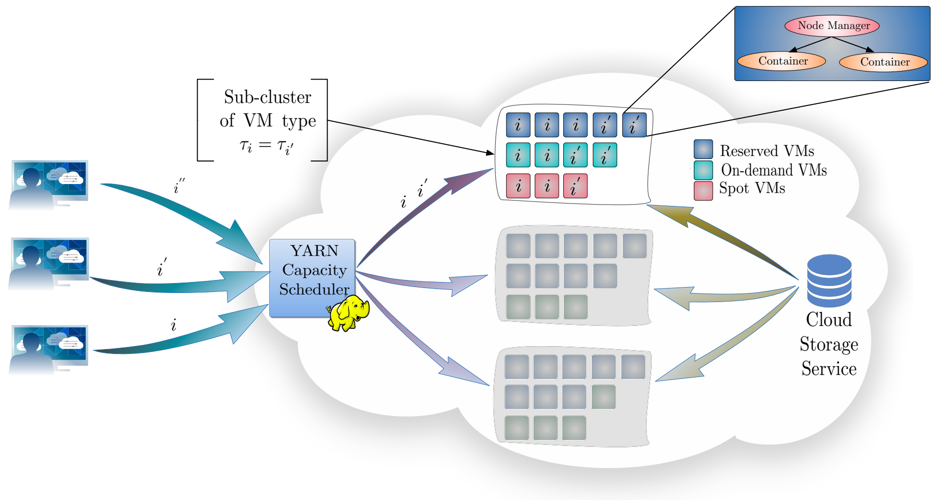


Figure 2. Reference system

Table 1
Model parameters

Param.	Definition
$\mathcal{C}$	Set of application classes
$\mathcal{V}$	Set of VM types
h_i	Number of concurrent users for class i
Z_i	Class i think time [ms]
D_i	Deadline associated to applications of class i [ms]
η_i	Maximum percentage of spot VMs allowed to class i
m_i	Class i YARN container memory size [GB]
v_i	Class i YARN container number of vCPUs
M_j	Memory size for a VM of type j [GB]
V_j	Number of vCPUs available within a VM of type j
ρ_j	Effective hourly price for reserved VMs of type j [€/h]
δ_j	Unit hourly price for on demand VMs of type j [€/h]
σ_j	Unit hourly price for spot VMs of type j [€/h]
R_{ij}	Number of reserved VMs of type j devoted to class i users

Table 2
Decision variables

Var.	Definition
ν_i	Number of VMs assigned for the execution of applications from class i
r_i	Number of reserved VMs booked for the execution of applications from class i
d_i	Number of on demand VMs assigned for the execution of applications from class i
s_i	Number of spot VMs assigned for the execution of applications from class i
c_i	Total number of YARN containers assigned to class i
x_{ij}	Binary variable equal to 1 if VM type j is assigned to application class i

problem. Furthermore, Table 1 reports a complete list of the parameters used in the models presented in the next sections, whilst Table 2 summarizes the decision variables.

4 Problem Formulation and Solution

This section presents the optimization models and techniques exploited by the D-SPACE4Cloud tool in order to rightsize the cluster deployment given a set of profiles characterizing the DIAs under study, the candidate VM types (possibly at different Cloud providers), and different pricing models. The resulting optimization model is the resource allocation problem presented in Section 4.1.

The first issue D-SPACE4Cloud has to tackle is to quickly, yet with an acceptable degree of accuracy, estimate the completion times and operational costs of the cluster: to this end, within the mathematical programming formulation of the problem ML models are exploited for the assessment of application execution times. In this way, it is possible to swiftly explore several plausible configurations and point out the most cost-effective among the feasible ones. Afterwards, the required resource configuration can be fine-tuned using more accurate, though more time-consuming and computationally demanding, simulations to obtain a precise prediction of the expected running time.

According to the previous considerations, the first step in the optimization procedure consists in determining the most cost-effective resource type for each application, based on their price and the expected performance. The mathematical programming models that allow to identify such an initial solution are discussed in Section 4.2. Finally, the algorithm adopted to efficiently tackle the resource provisioning problem is described in Section 4.3.

4.1 Optimization Model

The Big Data cluster resource provisioning problem can be formalized through the following mathematical programming formulation:

$$\min_{\mathbf{x},\boldsymbol{\nu},\mathbf{r},\mathbf{d},\mathbf{s}} \quad \sum_{i \in \mathcal{C}} \left(\rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i \right) \tag{P1a}$$

subject to:

$$\sum_{j \in \mathcal{V}} x_{ij} = 1, \quad \forall i \in \mathcal{C}, \tag{P1b}$$

$$R_{i,\tau_i} = \sum_{j \in \mathcal{V}} R_{ij} x_{ij}, \quad \forall i \in \mathcal{C}, \tag{P1c}$$

$$\rho_{\tau_i} = \sum_{j \in \mathcal{V}} \rho_j x_{ij}, \quad \forall i \in \mathcal{C}, \tag{P1d}$$

$$\delta_{\tau_i} = \sum_{j \in \mathcal{V}} \delta_j x_{ij}, \quad \forall i \in \mathcal{C}, \tag{P1e}$$

$$\sigma_{\tau_i} = \sum_{j \in \mathcal{V}} \sigma_j x_{ij}, \quad \forall i \in \mathcal{C}, \tag{P1f}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in \mathcal{C}, \forall j \in \mathcal{V}, \tag{P1g}$$

$$(\boldsymbol{\nu}, \mathbf{r}, \mathbf{d}, \mathbf{s}) \in \arg\min \sum_{i \in \mathcal{C}} \left(\rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i \right) \tag{P1h}$$

subject to:

$$r_i \leq R_{i,\tau_i}, \quad \forall i \in \mathcal{C}, \tag{P1i}$$

$$s_i \leq \frac{\eta_i}{1 - \eta_i} \left(r_i + d_i \right), \quad \forall i \in \mathcal{C}, \tag{P1j}$$

$$\nu_i = r_i + d_i + s_i, \quad \forall i \in \mathcal{C}, \tag{P1k}$$

$$\mathcal{T}_i \left(\nu_i, h_i, Z_i, \tau_i \right) \leq D_i, \quad \forall i \in \mathcal{C}, \tag{P1l}$$

$$\nu_i \in \mathbb{N}, \quad \forall i \in \mathcal{C}, \tag{P1m}$$

$$r_i \in \mathbb{N}, \quad \forall i \in \mathcal{C}, \tag{P1n}$$

$$d_i \in \mathbb{N}, \quad \forall i \in \mathcal{C}, \tag{P1o}$$

$$s_i \in \mathbb{N}, \quad \forall i \in \mathcal{C}. \tag{P1p}$$

Problem (P1) is presented as a bi-level resource allocation problem where the outer objective function (P1a) considers running costs. For each application class the logical variable x_{ij} is set to 1 if the VM type j is assigned to application class i, 0 otherwise. Constraints (P1b) enforce that only $x_{i,\tau_i} = 1$, thus determining the optimal VM type τ_i for application class i. Hence the following constraints, ranging from (P1c) to (P1f), pick the values for the inner problem parameters.

The inner objective function (P1h) has the same expression as (P1a), but in this case the prices ρ_{τ_i}, δ_{τ_i}, and σ_{τ_i} are fixed, as they have been chosen at the upper level. Constraints (P1i) bound the number of reserved VMs that can be concurrently started according to the contracts in place with the IaaS provider. The subsequent constraints (P1j) enforce that spot instances do not exceed a fraction η_i of the total assigned VMs and constraints (P1k) add all the VMs available for class i, irrespective of the pricing model. Further, constraints (P1l) mandate to respect the deadlines D_i. In the end, all the remaining decision variables are taken from the natural numbers set, according to their interpretation.

4.2 Identifying an Initial Solution

The presented formulation of Problem (P1) is particularly difficult to tackle, as it is a MINLP problem, possibly nonconvex, depending on $\mathcal{T}_i$. According to the literature about complexity theory [29], integer programming problems belong to the NP-hard class, hence the same applies to (P1). However, since there is no constraint linking variables belonging to different application classes, the general formulation can be split into several smaller and independent problems, one per class $i \in \mathcal{C}$:

$$\min_{c_i, r_i, d_i, s_i} \quad \rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i \tag{P2a}$$

subject to:

$$r_i \leq R_{i,\tau_i}, \tag{P2b}$$

$$s_i \leq \frac{\eta_i}{1 - \eta_i} \left(r_i + d_i \right), \tag{P2c}$$

$$m_i c_i \leq M_{\tau_i} \left(r_i + d_i + s_i \right), \tag{P2d}$$

$$v_i c_i \leq V_{\tau_i} \left(r_i + d_i + s_i \right), \tag{P2e}$$

$$\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} \leq D_i, \tag{P2f}$$

$$c_i \in \mathbb{N}, \tag{P2g}$$

$$r_i \in \mathbb{N}, \tag{P2h}$$

$$d_i \in \mathbb{N}, \tag{P2i}$$

$$s_i \in \mathbb{N}. \tag{P2j}$$

Problem (P2) drops ν_i thanks to constraints (P1k). Moreover, (P1l) becomes the system of constraints (P2d)–(P2f). Specifically, (P2d) and (P2e) ensure that the capacity associated with the overall number of containers, c_i, namely, $m_i c_i$ and $v_i c_i$, is consistent with nodes capacity, in terms of both vCPUs (V_{τ_i}) and memory (M_{τ_i}). As regards constraint (P2f), the presented inequality is a model of the average execution time, function of the concurrency level and the available containers, among other features, used to enforce that the completion time meets the arranged deadline. Specifically, it is the result of a ML process to get a first order approximation of the execution time of Hadoop and Spark jobs in Cloud clusters. More in details, building upon [30], which compares linear regression, Gaussian support vector regression (SVR), polynomial SVR with degree ranging between 2 and 6, and linear SVR, this paper adopts a model learned with linear SVR, following [28]. This is due to the fact that SVR with other kinds of kernel fares worse than with the linear one, whilst plain linear regression requires specific data cleaning to avoid perfect multicollinearity in the design matrix, thus making it harder to apply in the greatest generality. In order to select a relevant feature set, a good starting point is generalizing the analytical bounds for MapReduce clusters proposed in [15]. This approach suggests a diverse collection of features including the number of tasks in each map or reduce phase, or stage in the case of Spark applications, average and maximum values of task execution times, average and maximum shuffling times, dataset size, as well as the number of available containers, for which the reciprocals are considered. The $\mathcal{T}_i$ function in (2) becomes, then:

$$T_i = \sum_{k \in \mathcal{F}_i} w_{i,\tau_i,k} x_{ik}, \tag{4}$$

where $\mathcal{F}_i$ is the set of features associated to application i, x_{ik} the values assumed by such features, and $w_{i,\tau_i,k}$ the coefficients learned via SVR. These depend also on the chosen VM type, due to the influence of its computational capabilities on the observed performance. The above exemplified measures appear within $\mathbf{x}_i$. Since most of these features characterize the application class, but cannot be controlled, equation (P2f) collapses all but h_i and c_i, with the corresponding coefficients, into a single constant term, χ^0_{i,τ_i}, that is the linear combination of the feature values with the SVR-derived weights.

Problem (P2) can be reasonably relaxed to a continuous formulation as in other literature approaches (see, e.g., [31]). Furthermore, the problem can be additionally simplified with a couple of simple algebraic transformations.

First, constraints (P2d) and (P2e) share the same basic structure and are alternative, hence in every feasible solution at most one can be active. Building upon this consideration, it is possible to reformulate them as a single constraint, the most stringent:

$$c_i \leq \alpha_i \left(r_i + d_i + s_i \right), \text{ where } \alpha_i \triangleq \min \left\{ \frac{M_{\tau_i}}{m_i}, \frac{V_{\tau_i}}{v_i} \right\}. \tag{5}$$

Moreover, given the total number of VMs needed to support the required workload, it is trivial to determine the optimal instance mix using dominance considerations. Indeed, since $\sigma_{\tau_i} < \rho_{\tau_i} < \delta_{\tau_i}$, spot VMs are selected first, but respecting the constraint (P2c), then it is the turn of reserved ones, whose number is bounded by R_{i,τ_i} and, at last, on demand instances cover the still unheeded computational needs. Moving from this consideration, it is possible to reduce the problem to a formulation that involves only the number of containers, c_i, and the overall number of VMs, ν_i, as exposed below:

$$\min_{c_i, \nu_i} \nu_i \tag{P3a}$$

subject to:

$$c_i \leq \alpha_i \nu_i, \tag{P3b}$$

$$\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} \leq D_i, \tag{P3c}$$

$$c_i \geq 0, \tag{P3d}$$

$$\nu_i \geq 0. \tag{P3e}$$

The continuous relaxation makes it possible to apply the Karush-Kuhn-Tucker conditions, which are necessary and sufficient for optimality due to the regularity of Problem (P3), thanks to Slater's constraint qualification: (P3c) is the only nonlinear constraint and is convex in the domain, which in turn contains an internal point. Notice that, in this way, it is possible to analytically obtain the optimum of all the instances of Problem (P3), one per class $i \in \mathcal{C}$, as proven in Theorem 4.1.

Theorem 4.1. *The optimal solution of Problem* (P3) *is:*

$$c_i = \frac{\chi^c_{i,\tau_i}}{D_i - \chi^h_{i,\tau_i} h_i - \chi^0_{i,\tau_i}}, \tag{6a}$$

$$\nu_i = \frac{c_i}{\alpha_i} = \frac{1}{\alpha_i} \frac{\chi^c_{i,\tau_i}}{D_i - \chi^h_{i,\tau_i} h_i - \chi^0_{i,\tau_i}}. \tag{6b}$$

Proof. The Lagrangian of Problem (P3) is given by:

$$\begin{aligned} \mathcal{L}\left(c_i, \nu_i\right) = {} & \nu_i + \lambda_\alpha \left(c_i - \alpha_i \nu_i\right) + \\ & + \lambda_\chi \left(\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} - D_i \right) + \\ & - \lambda_c c_i - \lambda_\nu \nu_i \end{aligned} \tag{7}$$

and stationarity conditions lead to:

$$\frac{\partial \mathcal{L}}{\partial \nu_i} = 1 - \alpha_i \lambda_\alpha - \lambda_\nu = 0, \tag{8a}$$

$$\frac{\partial \mathcal{L}}{\partial c_i} = \lambda_\alpha - \lambda_\chi \chi^c_{i,\tau_i} \frac{1}{c_i^2} - \lambda_c = 0, \tag{8b}$$

while complementary slackness conditions are:

$$\lambda_\alpha \left(c_i - \alpha_i \nu_i\right) = 0, \quad \lambda_\alpha \geq 0, \tag{9a}$$

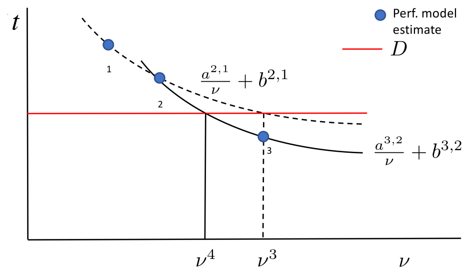


Figure 3. Hyperbolic jump

$$\lambda_\chi \left(\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} - D_i \right) = 0, \quad \lambda_\chi \geq 0, \tag{9b}$$

$$\lambda_c c_i = 0, \quad \lambda_c \geq 0, \tag{9c}$$

$$\lambda_\nu \nu_i = 0, \quad \lambda_\nu \geq 0. \tag{9d}$$

Constraint (P3c) requires $c_i > 0$ and, thanks to (P3b), it also holds $\nu_i > 0$. Thus, $\lambda_c = 0$ and $\lambda_\nu = 0$. Now, equations (8a) and (8b) can be applied to obtain $\lambda_\alpha > 0$ and $\lambda_\chi > 0$. So constraints (P3c) and (P3d) are active in every optimal solution, whence descend (6a) and (6b). $\square$

Since Theorem 4.1 provides optima in closed form for Problem (P3), it is straightforward to repeat its algebraic solution for all the pairs class-VM of Problem (P1). The choice of the preferred VM type whereon to run each class is made via the comparison of all the relevant optimal values, selecting by inspection the minimum cost association of classes and VM types.

4.3 The Optimization Algorithm

The aim of this section is to provide a brief description of the optimization heuristic embedded in D-SPACE4Cloud. It efficiently explores the space of possible configurations, starting from the initial ones obtained via Theorem 4.1.

Since (P3c) is only a preliminary approximation, the very first step of the procedure is simulating the initial configuration in order to refine the prediction. This step, as well as all the subsequent ones, is executed in parallel since the original Problem (P1) has been split into independent subproblems. After checking the feasibility of the initial solution, the search algorithm begins the exploration incrementing the VM assignment whenever the solution results infeasible, or decreasing it to save on costs if the current configuration is already feasible.

In order to avoid one-VM steps, which might lead to a very slow convergence for the optimization procedure, particularly when dealing with large clusters, the optimization heuristic exploits the fact that the execution time of DIAs (as approximated by (P2f)) is inversely proportional to the allocated resources (see also [15], [22], [28]). Hence, at every iteration the application execution time is estimated as:

$$t_i = \frac{a_i}{\nu_i} + b_i, \tag{10}$$

where t_i is the execution time and ν_i the number of VMs, whilst a_i and b_i are obtained by fitting the hyperbola to the results of the previous steps. Hence, from the second search step on, a_i and b_i can be computed using the predicted

Algorithm 1 Search algorithm

```
Require: $\nu_i^0 \in \mathbb{N}$
 1: simulate $\nu_i^0$
 2: if $\nu_i^0$ is infeasible then
 3:     $\nu_i^1 \leftarrow \nu_i^0 + 1$
 4:     $l_i^1 \leftarrow \nu_i^0$
 5: else
 6:     $\nu_i^1 \leftarrow \nu_i^0 - 1$
 7:     $u_i^1 \leftarrow \nu_i^0$
 8: end if
 9: repeat $k \leftarrow 1, 2, \ldots$
10:     simulate $\nu_i^k$
11:     update bounds
12:     $\nu_i^{k+1} \leftarrow f\left(\nu_i^k, \nu_i^{k-1}\right)$
13:     check $\nu_i^{k+1}$ against bounds
14: until $u_i^k - l_i^k = 1$
15: return $u_i^k$
```

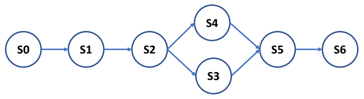
execution times returned by the performance simulators and the associated resource allocations. In this way, at every iteration k it is possible to have an educated guess on the number of VMs required to meet the deadline D_i, as depicted in Figure 3:

$$\nu_i^{k+1} = \frac{a_i^{k,k-1}}{D_i - b_i^{k,k-1}}. \tag{11}$$

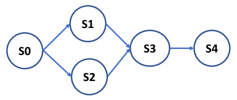
The proposed optimization algorithm aims at combining the convergence guarantees of dichotomic search with the fast exploration allowed by specific knowledge on system performance, such as equations (10) and (11). Each job class is optimized separately and in parallel as described in pseudo-code in Algorithm 1. First off, the initial solution ν_i^0, obtained as outlined in Section 4.2, is evaluated using the simulation model. Since equation (11) requires at least two points, the conditional at lines 2–8 provides a second point at one-VM distance and sets the initial one as lower or upper bound, according to its feasibility. Then the algorithm iteratively searches the state space performing simulations and keeping track of the interval enclosing the optimal solution. Every new step relies on the hyperbolic function, as shown at line 12.

As has already been mentioned, D-SPACE4Cloud mixes dichotomic search and domain knowledge about performance characteristics in order to exploit the best of both worlds. Fitting a hyperbola to previous results allows for speeding up the exploration by directing it where the system performance is expected to be reasonably close to the deadline imposed as constraint, yet the use of only the latest two simulations, dictated by convenience considerations, might hamper convergence with oscillations due to inaccuracies. This issue is addressed by recording the most resource hungry infeasible solution as lower bound, l_i^k, and the feasible configuration with fewest VMs as upper bound, u_i^k. Hence, at line 11, if ν_i^k turns out to be feasible, then it is assigned to u_i^k, otherwise to l_i^k. Furthermore, every new tentative configuration ν_i^{k+1} predicted at line 12 must belong to the open interval $\left(l_i^k, u_i^k\right)$ to be relevant: at line 13 the algorithm enforces this behavior, falling back to the mid point when this property does not hold.

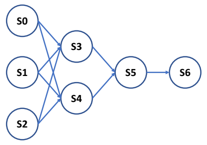
Now, given the monotonic dependency of execution times on the number of assigned computational nodes, the stopping criterion at line 14 guarantees that the returned configuration is the provably optimal solution of the inner, separate Problem (P2) for class i. In other words, the joint selection of the VM type and their number is NP-hard, but when the type of VM is fixed in the first phase, the discussed heuristic obtains the final solution for all classes in polynomial time.



(a) Q26



(b) Q52



(c) Q40

Figure 4. Spark queries DAGs

5 Experimental Analysis

This section presents the results of an extensive experimental campaign aimed at validating the proposed approach.[3] All these experiments have been performed on two Ubuntu 14.04 VMs hosted on a server equipped with an Intel Xeon E5530 2.40 GHz microprocessor. The first VM ran D-SPACE4Cloud and dagSim, with 8 virtual CPUs and 12 GB of RAM. The second one, instead, ran JMT 0.9.3, with 2 virtual CPUs and 2 GB of RAM.

The analysis is structured as follows: Section 5.1 describes the experiment settings, while Section 5.2 introduces Ernest [22], an alternative solution provided by the literature for Big Data cluster sizing, which is considered as reference. Section 5.3 validates the simulation models against the performance of real clusters, Section 5.4 presents a comparative study on outcomes obtained by varying the problem parameters, thus identifying the potential savings of the proposed approach. Section 5.5 is devoted to assess the quality of solutions returned by D-SPACE4Cloud and to provide a quantitative comparison with Ernest. Finally, the scalability of the presented approach is studied in Section 5.6.

5.1 Experimental Setup

To profile Big Data applications and compare with simulator results, real measures have been collected by running SQL queries on Apache Hive[4] on MapReduce and Apache Spark.[5] The industry standard TPC-DS[6] benchmark dataset generator has been exploited to create synthetic data at scale factors ranging from 250 GB to 1,000 GB.

s **??** and **??**, available in the Appendix, show the considered queries: R1, R3, Q26, Q40, and Q52. R1 and R3 are handcrafted so as to have exactly one map and one reduce stage when run on Hive, thus constituting examples of MapReduce jobs. On the other hand, Q26, Q40, and Q52 belong to the TPC-DS benchmark. These queries have been executed on SparkSQL, yielding the DAGs depicted in Figure 4.

3. Supporting data is available on Zenodo: https://doi.org/10.5281/zenodo.1299775.
4. https://hive.apache.org
5. https://spark.apache.org
6. www.tpc.org/tpcds/

Since profiles are meant to statistically summarize jobs' behavior, the profiling runs have been performed at least twenty times per query to reduce the effect of outliers and the overall variance. Properly parsing the logs allows to extract all the parameters needed to build the query profile as, for instance, average and maximum task execution times, number of tasks, etc. Profiling has been performed on Amazon EC2, by considering m4.xlarge instances, on Microsoft Azure, with A3, A4, D12v2, D13v2, or D14v2 VMs, and on PICO,[7] the Big Data cluster provided by CINECA, the Italian supercomputing center. In addition, there is also a set of profiles obtained on an in-house cluster based on IBM POWER8 processors, available at Politecnico di Milano.

The cluster created in EC2 is composed of 30 computational nodes, for a total of 120 vCPUs hosting 240 YARN containers, whilst on PICO up to 120 cores configured to host one container per core have been used. As far as Azure is concerned, clusters of variable sizes have been provisioned, reaching up to 26 dual-core containers. The POWER8 deployment includes four worker VMs with 11 cores and 60 GB RAM each, plus one similarly configured master node, for a total of 44 CPUs available for computation. Spark executors have been configured with 2 cores and 4 GB RAM, while 8 GB have been allocated to the driver. In the EC2 case, every container has 2 GB RAM and on Cineca 6 GB, whereas on Azure containers feature 2 GB for A3 machines, 8 GB for A4 and D12v2, 40 GB for D13v2, and 90 GB for D14v2. Along with the profiles, it was also possible to collect lists of task execution times to feed the replayer in JMT service centers or dagSim stages. In the end, the different VM types characteristics have been recorded.

Our previous work [18] shows that GreatSPN, a tool based on SPNs, can reach a slightly higher accuracy than JMT at the expense of quite longer simulation times, thus here we do not consider it as simulator to achieve shorter optimization times. The cited work also highlights that MapReduce and Spark stages tend to follow Erlang distributions, whose coefficient of variation is small.

5.2 Comparison with an Alternative Method

With the aim of proving the soundness of the approach implemented in our tool, we have identified a state-of-the-art solution, Hemingway [21], for a problem that bears many similarities with the one addressed in this work. Hemingway harnesses ML performance models and Bayesian optimization to converge to the optimal configuration for recurrent jobs. A recurrent job is an application always executed within the same time limit. Compared to the method discussed in this paper, Hemingway has some serious limitations that prevent its direct application to our problem: i) it only considers single user scenarios, and ii) it must be applied fixing the application and service level agreements (SLAs). Nevertheless, the underlying performance model, obtained by relying on the Ernest [22] framework, can still be used to create an alternative algorithm suitable for the comparison. However, since the model itself is valid only for the single-user case, this is the scenario considered in the following comparisons.

Ernest's authors propose to apply nonnegative least squares fitting for learning a performance model of the form:

7. www.hpc.cineca.it/hardware/pico

$$t = \theta_0 + \theta_1 \frac{s}{m} + \theta_2 \log(m) + \theta_3 m, \tag{12}$$

where t is the execution time, s the input data size, and m the number of allocated machines. Each of the variable terms serves the purpose of modeling a specific common communication/computation pattern implemented in distributed applications: namely, the term in θ_1 relates to data-parallel processing—for example map stages, whilst the logarithmic term models tree-shaped aggregation—typical of reduce stages, and $\theta_3 m$ highlights communication targeting only one sink node—such as the collect action in Spark.

Further, Ernest's authors propose a methodology to train the model based on optimal design of experiment, yet to maximize the fairness of the comparison, we used the same training dataset of D-SPACE4Cloud's internal SVR model, which is very fine-grained and includes a very large set of experiments on real clusters where the system configuration varies with two-core steps for every platform. In this way, model learning happens with an abundance of data, so as not to impair the baseline.

After applying the learned model to predict performance across the whole range of possible machine allocations (by increasing m with step 1 starting from 1 to a maximum value such that the prediction obtained from (12) is lower than the target deadline D_i, for each candidate VM type), the optimal configuration is obtained by inspection, that is taking the minimum cost configuration whose predicted time is feasible, i.e., shorter than the imposed deadline. Adopting this approach, it is possible to determine the global optimum associated with the (Ernest-based) baseline model. Similarly, using measurements on the real system it is possible to establish by inspection the minimum cost configuration that meets a given deadline.

5.3 Simulation Models Validation

To start off with, here are presented results for the validation of the different performance models, parametrized using information and measurements from the real systems they aim at mimicking. The simulated model metrics (obtained through dagSim or Ernest's ML model) are compared to measurements, in order to assess accuracy. Specifically, the quality index used in the comparison is the signed relative error on the prediction of execution times, defined as follows:

$$\vartheta = \frac{\tau - T}{T} \tag{13}$$

where τ is the execution time obtained through performance prediction, whilst T is the average measurement over twenty runs. Such a definition allows not only to quantify the relative error on execution times, but also to identify cases where the predicted time is smaller than the actual one, thus leading to deadline misses. Indeed, if $\vartheta < 0$ then the prediction is not conservative.

The Appendix details the experimental campaign for performance models validation. These experiments are single user scenarios with a query repeatedly running on a dedicated cluster and a 10 s average think time. Every table details experiments with the relevant query, the allocated cores, the overall number of tasks to complete, as well as the mean

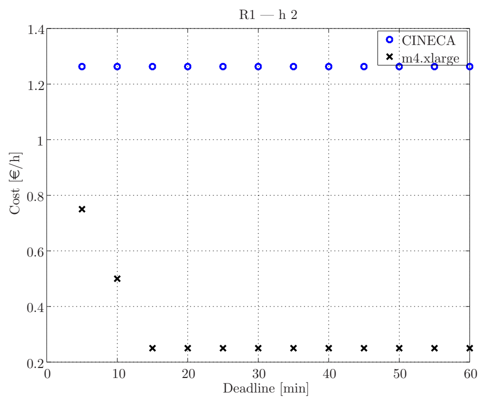


Figure 5. Query R1, two concurrent users

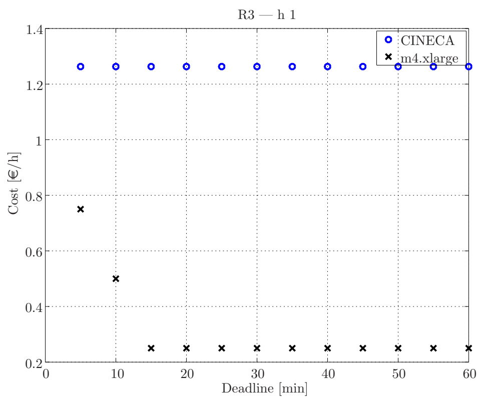


Figure 6. Query R3, one concurrent user

execution time T across twenty runs, the corresponding standard deviation s_T, obtained as square root of the unbiased estimator for variance, and the errors obtained with dagSim (ϑ_{d}) and Ernest (ϑ_{e}). s **??** and **??** report the results for dagSim models on the 500 GB dataset, with two different Azure VM types. Since all these experiments have a single user, it is possible to compare dagSim's results to the baseline model described in Section 5.2. The worst case error is −19.02% for dagSim and 18.53% when using Ernest, while on average errors settle at 3.05% for the former and 5.05% for the latter. Analogously, s **??**, **??**, **??** and **??** report the comparison of dagSim and Ernest with the deployment on IBM POWER8 machines and across all scale factors, considering Q40. Here dagSim reaches a worst case error of 6.01% against the 38.33% scored by Ernest, whilst the average errors are, respectively, 1.02% and 9.46%. Notice that, in this case, the model in (12) is not accurate enough, hence the further term $\theta_4 s^2 m^{-1}$ is added to improve accuracy. This particular term is mentioned in the original publication [22] as another contribution that might be needed for some workloads. In general, it is possible to observe in all the tables that dagSim mostly produces errors on the same scale as the measured standard deviation, fact that does not always apply to the baseline model. Similar results for QNs (14.13%) and SPNs (9.08%) models, considering multiple users, can be found in [18].

5.4 Scenario-based Experiments

The optimization approach described in Section 4 needs to be validated to assess its capability of catching realistic behaviors as one can reasonably expect of the system under analysis. This property is tested with a set of assessment instances where all the problem parameters but one are fixed, so as to verify that solutions follow an intuitive evolution.

The main axes governing performance in Hadoop or Spark clusters hosted on public Clouds are the level of concurrency and the deadlines. In the first case, increasing h_i and fixing all the remaining parameters, it is reasonable to expect a need for more VMs to support the higher workload, eventually leading to increased leasing costs. If the deadlines D_i tighten keeping the h_i unchanged, however, the optimal configuration should show increased costs anew as the system should require a higher parallelism to shrink execution times.

For the sake of clarity, in this section are reported single-class experiments in order to ensure an easier interpretation

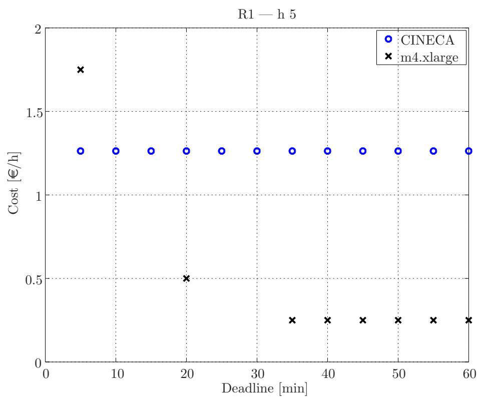


Figure 7. Query R1, five concurrent users

of the results. Figures 5, 6 and 7 show the solutions obtained with the 250 GB dataset on MapReduce queries using the JMT simulator. The average running time for each experiment is around two hours. All the mentioned figures show the cost in €/h plotted against decreasing deadlines in minutes for both the real VM types considered as candidate, namely, *CINECA*, the 20-core node machine available on PICO, and *m4.xlarge*, the 4-core instance available on Amazon AWS. In Figures 5 and 6 the expected cost increase due to tightening deadlines is apparent for both query R1 and R3. Further, in both cases it is cheaper to provision a cluster consisting of the smaller Amazon-offered instances, regardless the deadlines. Interestingly, R1 shows a different behavior when the required concurrency level increases from two to five users. Figure 7 shows that, as the deadlines tighten, it is possible to identify a region where executing the workload on larger VMs becomes more economic, with a 27.8% saving.

Figures 8 and 9 show the behavior of several Spark runs on the 500 GB dataset using the dagSim simulator. Q40 with ten users exhibits a straightforward behavior: D13v2 instances always lead to cheaper deployments. In order to quantify monetary savings, the ratio of the difference between costs over the second cheapest alternative has been computed. With this metric, D13v2 yields an average percentage saving around 23.1% for Q40, hence this VM type proves to be the cheapest choice by a reasonable margin. Single-user Q52, conversely, provides a more varied scenario. As shown in Figure 10 for clarity, two VM types, namely, A3 and D12v2, alternate as the cheapest deployment when the deadline varies. This

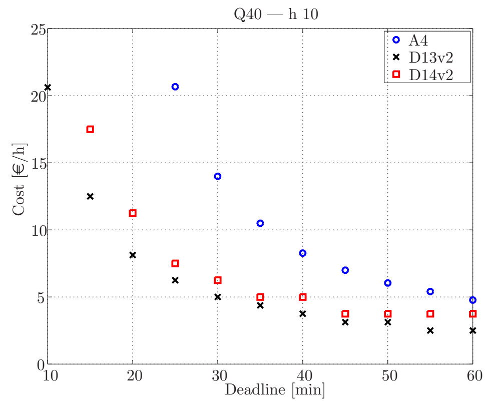


Figure 8. Query Q40, ten users

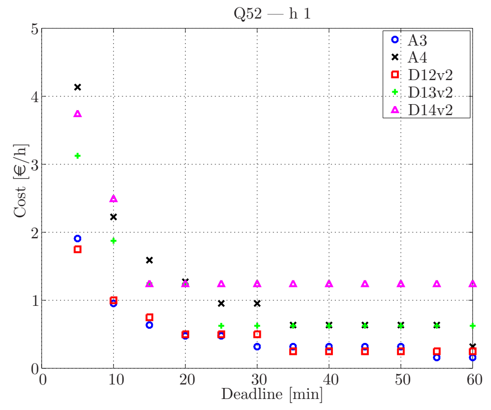


Figure 9. Query Q52, single user

figure reports the percentage savings gained by the cheapest deployment choice in comparison to the second cheapest. By identifying the proper alternative, it is possible to obtain an average saving around 19.3% over the considered range of deadlines, whereas the maximum saving is about 36.4%. Finally, Figure 11 shows the results of a multi-user experiment over the 500 GB dataset. All the other parameters fixed, among them query Q26 and a deadline of 20 minutes, the required concurrency level varies between 1 and 10 users with step 1. The figure, accordingly, plots hourly operational costs against concurrency levels. In this experiment the D12v2 instances prove in every case to be the better choice, with an

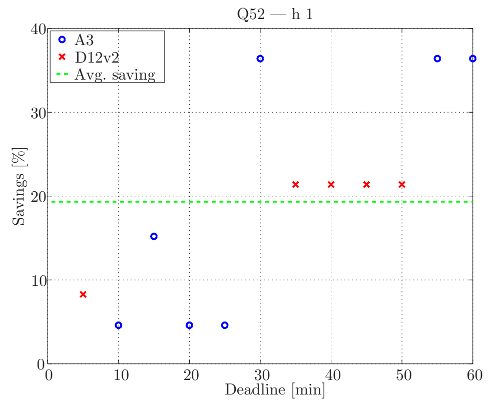


Figure 10. Query Q52, single user, savings

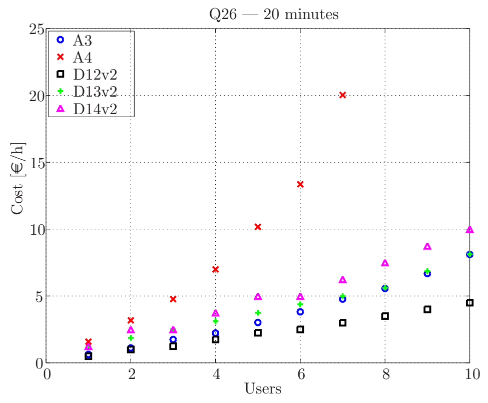


Figure 11. Query Q26, multi-user

average 30.0% saving in comparison to the second cheapest deployment.

In terms of execution times, D-SPACE4Cloud has carried out the whole optimization procedure for Spark experiments within minutes. All the running times are in the range $[24, 560]$ s, with an average of 125.98 s. In these cases the search algorithm runs much faster due to the performance gain allowed by dagSim, which has been used for the Q queries.

Overall, these results provide a strong point in favor of the optimization procedure implemented in D-SPACE4Cloud, as they prove that making the right choice for deployment can lead to substantial savings throughout the application life cycle. Suffice it to think that Microsoft Azure suggests VMs of the D11–15v2 range for memory intensive applications, such as analytics on Spark or distributed databases, whilst these results highlight that, under many circumstances, even the most basic offerings in the A0–4 range can satisfy QoS constraints with a competitive budget allocation. On top of this, D-SPACE4Cloud can also determine the optimal number of VMs to use in order to meet the QoS requirements, which is a nontrivial decision left to users.

5.5 Solution Validation in a Real Cluster Setting

A further experiment is aimed at assessing the quality of the optimized solution obtained using D-SPACE4Cloud. Given a query and a deadline to meet, the considered quality index is the relative error on allocated cores. Formally:

$$\varepsilon = \frac{c^{\mathrm{p}} - c^{\mathrm{r}}}{c^{\mathrm{r}}}, \qquad (14)$$

where c^{p} is the optimal number of cores returned by D-SPACE4Cloud and c^{r} the actual optimum obtained by inspection from measurements on the real system. With this definition, when the optimizer assigns too scarce resources the error is negative.

As in Section 5.3, here is a comparison of the results obtained via D-SPACE4Cloud and dagSim (ε_{d}) against Ernest [22] (ε_{e}). As previously discussed, since Ernest provides a formula, it is possible to obtain the minimum cores feasible configuration by inspection.

Table 3 considers six cases, varying deadline and query, and lists the optimal resource assignment to meet the QoS constraints on D12v2 instances with a 500 GB scale factor.

Table 3
Optimizer single class validation, D12v2

Query	D [ms]	c^{r}	ε_{d} [%]	ε_{e} [%]
Q26	180,000	48	0.00	16.67
Q52	180,000	48	0.00	0.00
Q26	240,000	40	−10.00	0.00
Q52	240,000	40	−10.00	−10.00
Q26	360,000	24	0.00	16.67
Q52	360,000	24	0.00	0.00

Table 4
Optimizer single class validation, IBM POWER8

Query	D [ms]	c^{r}	ε_{d} [%]	ε_{e} [%]	$\varepsilon_{\mathrm{e'}}$ [%]
Q40	2,537,015	6	0.00	0.00	33.33
Q40	1,905,821	8	0.00	−25.00	25.00
Q40	1,566,992	10	0.00	−20.00	20.00
Q40	1,337,868	12	0.00	−16.67	16.67
Q40	1,165,666	14	0.00	−28.57	28.57
Q40	1,044,557	16	0.00	−25.00	25.00
Q40	964,752	18	0.00	−22.22	22.22
Q40	902,173	20	0.00	−30.00	10.00
Q40	854,692	22	0.00	−27.27	9.09
Q40	812,298	24	0.00	−33.33	8.33
Q40	784,445	26	−7.69	−38.46	0.00
Q40	763,447	28	−7.14	−35.71	0.00
Q40	743,283	30	−6.67	−40.00	−6.67
Q40	726,324	32	−6.25	−43.75	−6.25
Q40	718,641	34	−11.76	−47.06	−11.76
Q40	696,641	36	−5.56	−50.00	−16.67
Q40	687,299	38	−10.53	−47.37	−15.79
Q40	664,080	40	0.00	−50.00	−20.00
Q40	662,526	42	−4.76	−52.38	−23.81
Q40	663,823	44	−9.09	−54.55	−27.27

Every row shows the relevant query and deadline, the optimal number of cores, and the percentage errors with both alternatives. The two methods prove accurate, with relative errors that always remain below 20%, on average 3.33% for D-SPACE4Cloud and 7.22% for Ernest.

Table 4, on the other hand, reports an extensive comparison over the full range of experiments run on the internal cluster based on IBM POWER8 processors for query Q40. In particular, the table shows results obtained at a 1000 GB scale factor, with deadlines set at the mean execution times (a.k.a. response times [27]) measured on the real system. In this case, Ernest with the basic feature set widely underestimates the optimal resource allocation across the board. In order to obtain a fair comparison, the column marked $\varepsilon_{\mathrm{e'}}$ lists the errors yielded by the Ernest model tweaked using the additional feature provided by the squared scale factor divided by the number of cores, at which the authors of Ernest hint in their paper, stating that for some workloads it is useful. This analysis shows that the proposed approach exploiting D-SPACE4Cloud and dagSim attains a better accuracy than the baseline, with an average relative error of 3.47% instead of the 16.32% obtained by Ernest with the additional feature, or 34.37% when this is not used. The worst value for the error is also lower, −11.76% against 33.33% obtained by the improved Ernest model.

At last, Table 5 presents the results for a multi-class instance on D14v2 VMs. Three different queries subject to the same deadline are executed on a shared cluster of three

Table 5
Optimizer multi-class validation, D14v2

Query	D [ms]	Cores	R [ms]	ζ [%]
Q26	720,000	16	533,731	25.87
Q40	720,000	16	530,122	26.37
Q52	720,000	16	562,625	21.86

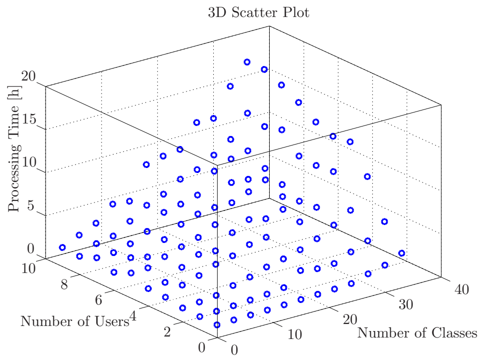


Figure 12. Execution time for varying number of classes and users

worker nodes (as mandated by the optimal solution returned by D-SPACE4Cloud), for a total of 48 cores under a work conserving scheduling policy. Under this scheduling policy, resources can be borrowed from other classes during their idle time, so the accuracy metric adopted in this scenario is the relative error of measured execution times against the deadline (ζ): the worst case result is 26.37%, with an average of 24.70%.

5.6 Scalability Analysis

This section reports the analysis over the times taken to obtain the optimized solution for instances of increasing size, both in terms of number of classes and aggregated user count. All these runs harness dagSim as simulator and Azure D14v2 VMs as target deployment.

The experiment considers three different queries, namely, Q26, Q40, and Q52, varying the deadline between five minutes and one hour with a five-minute stride, to obtain 12 sets of three distinct classes. Thus, there are Q26, Q40, and Q52 with deadline 60 minutes, then the same three queries all with a deadline of 55 minutes, and so on. The instances are then created cumulatively joining the three-class sets following decreasing deadlines. For example, the configuration with three classes has $D = 60$ minutes, the second instance with six classes collects $D \in \{60, 55\}$ minutes, the third one adds $D = 50$, and so forth. This test instance generation has been repeated with a required level of concurrency ranging from 1 to 10 users, but without mixing classes with different h_i, that is in any given instance $\forall i \in \mathcal{C},\ h_i = \bar{h}$. In this way, a total of 120 different test instances have been generated with varying number of classes and overall concurrent users: classes range between 3 and 36, while the aggregate number of users from 3 to 360.

Figure 12 shows the results of this experiment. The plot represents the mean execution time of D-SPACE4Cloud at every step. The number of users is per class, thus, for example, the configuration (24, 4) totals 96 users subdivided into 24 distinct classes with concurrency level 4. Overall, the average processing time ranges from around 40 minutes for three classes up to 12 hours for 36 classes. On the other hand, the best single instance, which needs only three minutes, is with three classes and one user each, while the longest running takes 16 hours and a half to optimize 36 classes of 9 users.

The reported results show how D-SPACE4Cloud can solve the capacity allocation problem for Cloud systems in less than one day, which is a reasonable time span for design time considerations. Furthermore, current best practices discourage hosting as many as 36 application classes and 360 users on a single shared cluster, hence several of the considered instances could be considered one order of magnitude larger than nowadays production environments.

6 Related Work

Capacity planning and architecture design space exploration are important problems addressed in the literature [32], [33]. High level models and tools to support software architects (see, e.g., Palladio Component Model and its Palladio Bench and PerOpteryx design environment [34], [35], or stochastic process algebra and the PEPA Eclipse plugin [36], [37]) have been proposed to identify optimized configurations given a set of QoS requirements for enterprise Web-based systems, but unfortunately they do not support Cloud-specific abstractions or directly address the problem of deriving an optimized Cloud and Big Data cluster configuration [38].

Sun et al. [39] propose ROAR, a tool to automatically benchmark Cloud resources in order to deploy multi-tiered Web applications in the most cost-effective way, yet avoiding QoS violations. Another approach is proposed in [40], where the authors use integer linear programming (ILP) to determine the optimal placement of Web components in the Cloud, pursuing high availability. Their approach consistently shortens the overall downtime compared to the OpenStack default scheduler, which is not availability-aware. Frey et al. [41] consider the problem of deploying existing applications in the Cloud rather than on premises. In order to find out the most fitting configurations, they adopt a simulation-based genetic algorithm and derive a Pareto front, according to costs, response times, and SLA violations.

Techniques specifically devised for Big Data workloads belong to two major categories depending on whether they are applied at design or run time. In the following, related works are organized according to this criterion.

6.1 Design Time Approaches

The problem of job profiling and execution time estimation represents a common issue in the Big Data literature. Verma et al. [42] propose a framework for the profiling and performance prediction of Hadoop applications running on heterogeneous resources. An approach to this problem based on closed QNs is presented in [43]. This work is noteworthy as it explicitly considers contention and parallelism on compute nodes to evaluate the execution time of a MapReduce application. However, the weak spot of this approach is that it contemplates the map phase alone. Vianna et al. [44] have worked on a similar solution; however, the validation phase has been carried out considering a cluster dedicated to the execution of a single application at a time.

A novel modeling approach based on mean field analysis, able to provide fast approximate methods to predict the performance of Big Data systems, has been proposed in [45]. Machine learning black box models are becoming also popular to predict the performance of large scale business analytics systems. Ernest [22] is a representative example of these approaches. The authors use experiment design to collect a reduced number of training points. In the experimental analysis the authors use Amazon EC2 and evaluate the accuracy of the proposed approach using several ML algorithms that are part of Spark MLlib: their evaluation shows that the average prediction error is under 20%. As previously discussed, this work is used as baseline for the evaluation of our proposal. Alipourfard et al. [10] present CherryPick, a black box system that leverages Bayesian optimization to find near-optimal Cloud configurations that minimize Cloud usage costs for MapReduce and Spark applications. The authors' approach also guarantees application performance and limits the overhead for recurring Big Data analytics jobs, focusing the search to improve prediction accuracy of those configurations that are close to the best for a specific deadline.

Similar to our work, the authors in [46] provide a framework facing the problem of minimum costs provisioning of data analytics clusters in Cloud environments. The cost model includes resource costs and SLA penalties, which are proportional to the deadline violations. Execution times of queries are predicted through QN models, which introduce up to 70% percentage error, though. The minimum cost configuration is chased through two greedy hill climbing heuristics, which can identify heterogeneous clusters, yet no guarantees on the quality of the solution can be provided.

In [28] the authors investigate a mixed analytical/ML approach to predict the performance of MapReduce clusters, harnessing QNs to generate a knowledge base (KB) of synthetic data whereon a complementary SVR model is trained. The initial KB is then updated over time to incorporate real samples from the operational system. Such method has been recently extended to model also Spark and is the approach used for building the ML models discussed in Section 4. Zhang et al. [47] investigate the performance of MapReduce applications on Hadoop clusters in the Cloud. They consider a problem similar to the one faced in this work and provide a simulation-based framework for minimizing cluster infrastructural costs.

In [15] the ARIA framework is presented. This work focuses on clusters dedicated to single-user classes handled by the FIFO scheduler. The framework addresses the problem of calculating the most suitable amount of resources to allocate to map and reduce tasks in order to meet a user-defined due date for a certain application; the aim is to avoid as much as possible costs due to resource over-provisioning. The same authors, in a more recent work [4], provide a solution for optimizing the execution of a workload specified as a set of DAGs under the constraints of a global deadline or budget. Heterogeneous clusters with possible faulty nodes are considered as well.

6.2 Run Time Approaches

Big Data frameworks often require an intense tuning phase in order to exhibit their full potential. For this reason, Starfish, a self-tuning system for analytics on Hadoop, has been proposed [48]. In particular, Starfish collects some key run time information about applications execution with the aim of generating meaningful application profiles; such profiles are, in turn, the basic elements exploited for Hadoop automatic configuration processes. Furthermore, also the cluster sizing problem has been tackled and successfully solved exploiting the same tool [49]. More recently, Dalibard et al. [50] have presented BOAT, a gray box framework, which supports developers to build efficient auto-tuners for their systems, in situations where generic auto-tuners fail. BOAT is based on structured Bayesian optimization and has been used to support the performance tuning of Cassandra clusters and of GPU-based servers for neural network computation.

MapReduce scheduling is considered in [14]. The authors propose a tandem queue with overlapping phases to model the execution of the application and an efficient run time scheduling algorithm for the joint optimization of the map and copy/shuffle phases. The authors demonstrate the effectiveness of their approach comparing it with the offline generated optimal schedule. Phan et al. [51] recognize the inadequacy of Hadoop schedulers released at the time of their writing in properly handling completion time requirements. The work proposes to adapt some classical multiprocessor scheduling policies to the problem; in particular, two versions of the earliest deadline first heuristic are presented and proved to outperform off-the-shelf schedulers. A similar approach is proposed in [47], where the authors present a solution to manage clusters shared among Hadoop applications and more traditional Web systems. Another recent contribution addresses the scheduling of long-running applications in shared clusters [52]. The authors couple a task-based scheduler, to cope with short jobs, and an ILP formulation that takes care of scheduling long-running applications while minimizing costs and violations of placement constraints, thus achieving up to 32% shorter median execution time in comparison to a baseline constraint-aware scheduler. The authors in [53] provide a run time framework for the management of large Cloud infrastructures based on collaborative filtering and classification, which supports run time decision of a greedy scheduler. The overall goal is to maximize infrastructure utilization while minimizing resource contention, taking into account also resource heterogeneity. The same authors have extended their work in [54], supporting resource scale-out decisions (i.e., determining if more servers can be beneficial for an application) and server scale-up (i.e., predicting if more resources per server are beneficial) for Spark and Hadoop applications. The authors demonstrate that their framework can effectively manage large systems, improving significantly infrastructure utilization and application performance. Even if their collaborative filtering approach requires to gather little data from the running applications, it requires a significant effort to initially profile the baseline benchmarking applications used to predict the effects of, e.g., resource contention and scale up/out decisions at run time: the exhaustive profiling of 30 workload types running from 1 to 100 nodes.

In the streaming research field, Hochreiner et al. [55] propose a Platform for Elastic Stream Processing that aims for the run time management of a stream processing engine deployed on multiple Clouds. Their architecture makes use of an optimization problem to adapt allocated resources to incoming data volume while minimizing operational costs; in this way, they achieve a 20% monetary saving at the expense of 28% SLA violations. Heinze et al. [56] propose a different approach to stream elasticity. Building upon threshold-based scaling techniques, they devise a method to automatically optimize their parametrization, in order to minimize running costs without sacrificing QoS. The experiments show that they can obtain, on average, the same number of violations with a 19% cost saving compared to a fixed configuration baseline.

7 Conclusions

This paper proposes an effective tool for capacity planning of YARN managed Cloud clusters to support DIAs implemented as MapReduce or complex DAG-based Apache Spark applications. A MINLP formulation based on ML models has been developed, and its initial solution is iteratively improved by a sim-heuristic optimization algorithm able to accurately assess application performance under different conditions. In this way, the tool is able to achieve a favorable trade-off between prediction accuracy and running times.

A comprehensive experimental validation has proved how the tool is a valuable contribution towards supporting different application classes over heterogeneous resource types. Moreover, situations where choosing the best VM type is not trivial have been highlighted and discussed. In these cases, sticking to small instances and scaling out proves to be less economic than switching to more powerful VMs that call for a smaller number of replicas: the decreased replication factor compensates the increased unit price in a not obvious way. Unfortunately, this is not always true and making the right choice can lead to substantial savings throughout the application life cycle, up to 20–30% in comparison with the second best configuration. Finally, a comparison with an alternative method based on state of the art proposals demonstrates how our solution is able to identify more accurate solutions (with an average percentage error of about 3% on the number of cores with respect to 7–16%).

Future work will extend D-SPACE4Cloud to support the resource provisioning of continuous applications that integrate batch and streaming workloads. Moreover, the run time cluster management scenario will be addressed, with more stringent constraints on the optimization times that impose the use of different, less time-consuming performance prediction techniques.

Acknowledgments

The results of this work have been funded by the European Commission under the DICE H2020 research project (grant agreement no. 644869).

Spark experiments have been supported by Microsoft under the *Top Compsci University Azure Adoption* program.

References

[1] H. V. Jagadish, J. Gehrke, *et al.*, “Big Data and its technical challenges,” *Commun. ACM*, vol. 57, no. 7, pp. 86–94, 2014.

[2] IDC. (Apr. 14, 2017). “Worldwide semiannual Big Data and analytics spending guide,” [Online]. Available: www.idc.com/getdoc.jsp?containerId=prUS42371417 (visited on 09/05/2018).

[3] K.-H. Lee, Y.-J. Lee, *et al.*, "Parallel data processing with MapReduce: A survey," *ACM SIGMOD Rec.*, vol. 40, no. 4, pp. 11–20, 2012.

[4] Z. Zhang, L. Cherkasova, and B. T. Loo, "Exploiting Cloud heterogeneity to optimize performance and cost of MapReduce processing," *SIGMETRICS Perf. Eval. Review*, vol. 42, no. 4, pp. 38–50, 2015.

[5] C. Shanklin. (Jun. 21, 2014). "Benchmarking Apache Hive 13 for enterprise Hadoop," [Online]. Available: https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html (visited on 09/05/2018).

[6] K. Kambatla, G. Kollias, *et al.*, "Trends in Big Data analytics," *J. Parallel Distrib. Comput.*, vol. 74, no. 7, pp. 2561–2573, 2014.

[7] G. Anadiotis. "Real-time applications are going places," [Online]. Available: www.zdnet.com/article/real-time-applications-are-going-places/ (visited on 05/28/2018).

[8] M. A. Greene and K. Sreekanti. (Jan. 26, 2016). "Big Data in the enterprise: We need an "easy button" for Hadoop," [Online]. Available: www.oreilly.com/pub/e/3643 (visited on 09/05/2018).

[9] J. Gantz and D. Reinsel. (Dec. 2012). "The digital universe in 2020," [Online]. Available: www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf (visited on 09/05/2018).

[10] O. Alipourfard, H. H. Liu, *et al.*, "CherryPick: Adaptively unearthing the best Cloud configurations for Big Data analytics," in *14th Symp. Networked Systems Design and Implementation (NSDI 17)*, USENIX, 2017, pp. 469–482.

[11] J. Polo, D. Carrera, *et al.*, "Performance-driven task co-scheduling for MapReduce environments," in *Network Operations and Management Symp. (NOMS 10)*, IEEE, 2010. DOI: 10.1109/NOMS.2010.5488494.

[12] B. T. Rao and L. S. S. Reddy, "Survey on improved scheduling in Hadoop MapReduce in Cloud environments," 2012. arXiv: 1207.0780 [cs.DC].

[13] Z. Zhang, L. Cherkasova, *et al.*, "Automated profiling and resource management of Pig programs for meeting service level objectives," in *Proc. 9th Int'l Conf. Autonomic Computing (ICAC 12)*, 2012, pp. 53–62.

[14] M. Lin, L. Zhang, *et al.*, "Joint optimization of overlapping phases in MapReduce," *SIGMETRICS Perf. Eval. Review*, vol. 41, no. 3, pp. 16–18, 2013.

[15] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic resource inference and allocation for MapReduce environments," in *Proc. 8th ACM Int'l Conf. Autonomic Computing (ICAC 11)*, 2011, pp. 235–244.

[16] "Amazon EC2 pricing," [Online]. Available: https://aws.amazon.com/ec2/pricing/ (visited on 05/28/2018).

[17] "Hadoop MapReduce next generation — Capacity Scheduler," [Online]. Available: https://hortonworks.com/blog/benchmarking-apache-hive-13-enterprise-hadoop/ (visited on 05/28/2018).

[18] D. Ardagna, S. Bernardi, *et al.*, "Modeling performance of Hadoop applications: A journey from queueing networks to stochastic well formed nets," in *Algorithms and Architectures for Parallel Processing (ICA3PP 16)*, ser. LNCS, vol. 10048, 2016, pp. 599–613.

[19] A. Brito, D. Ardagna, *et al.*, "D3.4 EUBra-BIGSEA QoS infrastructure services intermediate version," Tech. Rep., 2017. [Online]. Available: www.eubra-bigsea.eu/sites/default/files/D3.4%20EUBra-BIGSEA%20QoS%20infrastructure%20services.pdf.

[20] M. Ciavotta, E. Gianniti, and D. Ardagna, "D-SPACE4Cloud: A design tool for Big Data applications," in *Algorithms and Architectures for Parallel Processing (ICA3PP 16)*, ser. LNCS, vol. 10048, 2016, pp. 614–629.

[21] X. Pan, S. Venkataraman, *et al.*, "Hemingway: Modeling distributed optimization algorithms," 2017. arXiv: 1702.05865 [cs.DC].

[22] S. Venkataraman, Z. Yang, *et al.*, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *13th Symp. Networked Systems Design and Implementation (NSDI 16)*, USENIX, 2016, pp. 363–378.

[23] G. Casale, D. Ardagna, *et al.*, "DICE: Quality-driven development of data-intensive Cloud applications," in *7th Int'l Workshop Modeling in Software Engineering (MiSE 15)*, 2015. DOI: 10.1109/MiSE.2015.21.

[24] M. Artac, T. Borovsak, *et al.*, "Model-driven continuous deployment for quality DevOps," in *Proc. 2nd Int'l Workshop Quality-Aware DevOps (QUDOS 16)*, ACM, 2016, pp. 40–41.

[25] M. Bertoli, G. Casale, and G. Serazzi, "JMT: Performance engineering tools for system modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 10–15, 2009.

[26] S. Baarir, M. Beccuti, *et al.*, "The GreatSPN tool: Recent enhancements," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 4–9, 2009.

[27] E. D. Lazowska, J. Zahorjan, *et al.*, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1984.

[28] E. Ataie, E. Gianniti, *et al.*, "A combined analytical modeling machine learning approach for performance prediction of MapReduce jobs in Cloud environment," in *18th Int'l Symp. Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 16)*, IEEE, 2016. DOI: 10.1109/SYNASC.2016.072.

[29] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1990.

[30] A. M. Rizzi, "Support vector regression model for BigData systems," 2016. arXiv: 1612.01458 [cs.DC].

[31] D. Ardagna, M. Ciavotta, and M. Passacantando, "Generalized Nash equilibria for the service provisioning problem in multi-Cloud systems," *IEEE Trans. Serv. Comput.*, vol. 10, no. 3, pp. 381–395, 2017.

[32] A. Aleti, B. Buhnova, *et al.*, "Software architecture optimization methods: A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, pp. 658–683, 2013.

[33] F. Brosig, P. Meier, *et al.*, "Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures," *IEEE Trans. Softw. Eng.*, vol. 41, no. 2, pp. 157–175, 2015.

[34] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance predic-

tion," *J. Systems and Software*, vol. 82, no. 1, pp. 3–22, 2009.

[35] A. Koziolek, H. Koziolek, and R. Reussner, "PerOpteryx: Automated application of tactics in multi-objective software architecture optimization," in *Proc. ACM Conf. Quality of Software Architectures (QoSA 11)*, 2011, pp. 33–42.

[36] M. Tribastone, S. Gilmore, and J. Hillston, "Scalable differential analysis of process algebra models," *IEEE Trans. Softw. Eng.*, vol. 38, no. 1, pp. 205–219, 2012.

[37] J. Hillston and S. Gilmore. (Feb. 16, 2017). "PEPA: Performance evaluation process algebra," [Online]. Available: www . dcs . ed . ac . uk / pepa / tools/ (visited on 09/05/2018).

[38] J. Kross and H. Krcmar, "Model-based performance evaluation of batch and stream applications for Big Data," in *Proc. IEEE 25th Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 17)*, 2017, pp. 80–86.

[39] Y. Sun, J. White, *et al.*, "ROAR: A QoS-oriented modeling framework for automated Cloud resource allocation and optimization," *J. Systems and Software*, vol. 116, pp. 146–161, 2016.

[40] M. Jammal, A. Kanso, and A. Shami, "High availability-aware optimization digest for applications deployment in Cloud," in *Int'l Conf. Communications (ICC 15)*, IEEE, 2015, pp. 6822–6828.

[41] S. Frey, F. Fittkau, and W. Hasselbring, "Search-based genetic optimization for deployment and reconfiguration of software in the Cloud," in *35th Int'l Conf. Software Engineering (ICSE 13)*, IEEE, 2013, pp. 512–521.

[42] A. Verma, L. Cherkasova, and R. H. Campbell, "Profiling and evaluating hardware choices for MapReduce environments: An application-aware approach," *Perform. Eval.*, vol. 79, pp. 328–344, 2014.

[43] S. Bardhan and D. A. Menascé, "Queuing network models to predict the completion time of the map phase of MapReduce jobs," in *Proc. Computer Measurement Group Int'l Conf. (CMG 12)*, 2012, pp. 146–153.

[44] E. Vianna, G. Comarela, *et al.*, "Analytical performance models for MapReduce workloads," *Int'l J. Parallel Programming*, vol. 41, no. 4, pp. 495–525, 2013.

[45] A. Castiglione, M. Gribaudo, *et al.*, "Exploiting mean field analysis to model performances of Big Data architectures," *Future Generation Computer Systems*, vol. 37, pp. 203–211, 2014.

[46] R. Mian, P. Martin, and J. L. Vazquez-Poletti, "Provisioning data analytic workloads in a Cloud," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1452–1458, 2013.

[47] W. Zhang, S. Rajasekaran, *et al.*, "Minimizing interference and maximizing progress for Hadoop virtual machines," *SIGMETRICS Perform. Eval. Review*, vol. 42, no. 4, pp. 62–71, 2015.

[48] H. Herodotou, H. Lim, *et al.*, "Starfish: A self-tuning system for Big Data analytics," in *Proc. Conf. Innovative Data Systems Research (CIDR 11)*, vol. 11, 2011, pp. 261–272.

[49] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *Proc. 2nd ACM Symp. Cloud Computing (SOCC 11)*, 2011.

[50] V. Dalibard, M. Schaarschmidt, and E. Yoneki, "BOAT: Building auto-tuners with structured Bayesian optimization," in *Proc 26th Int'l Conf. World Wide Web (WWW 17)*, 2017, pp. 479–488.

[51] L. T. X. Phan, Z. Zhang, *et al.*, "An empirical analysis of scheduling techniques for real-time Cloud-based data processing," in *Int'l Conf. Service-Oriented Computing and Applications (SOCA 11)*, IEEE, 2011. DOI: 10.1109/SOCA.2011.6166240.

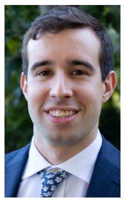
[52] P. Garefalakis, K. Karanasos, *et al.*, "MEDEA: Scheduling of long running applications in shared production clusters," in *Proc. Europ. Conf. Computer Systems (EuroSys 18)*, 2018. DOI: 10.1145/3190508.3190549.

[53] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," *SIGPLAN Not.*, vol. 48, no. 4, pp. 77–88, 2013.

[54] ——, "Quasar: Resource-efficient and QoS-aware cluster management," in *Proc. 19th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS 14)*, ACM, 2014, pp. 127–144.

[55] C. Hochreiner, M. Vögler, *et al.*, "Elastic stream processing for the Internet of Things," in *9th IEEE Int'l Conf. Cloud Computing (CLOUD 16)*, IEEE Computer Society, 2016, pp. 100–107.

[56] T. Heinze, L. Roediger, *et al.*, "Online parameter optimization for elastic data stream processing," in *Proc, of the Sixth ACM Symposium on Cloud Computing (SoCC)*, 2015, pp. 276–287.



Eugenio Gianniti received his Ph.D. degree in Computer Engineering in 2018 at Politecnico di Milano, Italy. His research interest lies mostly in optimization techniques for the resource management of DIAs hosted on Clouds, as well as in the performance modeling of such systems via both simulation and analytical methods.

Michele Ciavotta received the Ph.D. degree in Automation and Computer Science from the University of Roma Tre, Italy in 2008. He is researcher at the University of Milano-Bicocca since 2017. His research work focuses on modeling and optimization of highly constrained combinatorial problems mainly arising in the fields scheduling and resource management of distributed systems.

Danilo Ardagna is an Associate Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano, Milan, Italy. He received the Ph.D. degree in Computer Engineering from Politecnico di Milano in 2004. His work focuses on performance modeling of software systems and on the design, prototyping, and evaluation of optimization algorithms for resource management and planning of Cloud and Big Data systems.