D-SPACE4Cloud: Towards Quality-Aware Data Intensive 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 claiming 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 novel 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. We propose 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 resource configurations, seeking the minimum cost deployment that satisfies predefined quality of service constraints. The validity and relevance of the proposed solution has been thoroughly validated in a vast experimental campaign including 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 Big Data until recently concerned highly experimental projects, but its market will grow from $130.1 billion in 2016 to $203 billion in 2020, with a compound annual growth rate of 11.9%, with banking and manufacturing industries leading the investment market [2]. Big Data applications offer many business opportunities that stretch across industries, especially to enhance performance, as in the case of recommendation systems. In addition, data intensive applications (DIAs) can also help governments in obtaining accurate predictions, e.g., quality weather forecasts to prevent natural disasters and the development of appropriate policies to improve the population life quality. To corroborate these considerations, notice that the Obama government announced $200 million worth of investment to boost Big Data related industries and positioned this strategy into the national agenda in 2012.

One of the pillars on which the Big Data revolution is based is the MapReduce paradigm, which has allowed for 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's success has been planetary; it 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, suddenly, a whole world of opportunities.

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 arose. 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 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 and that, according to analysts, is expected to dominate the market for the next decade [7].

In spite of all the fuss around Big Data technologies, it is still undeniably true that fully embracing them is a very complex process. Many efforts have been made to make this technology 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]. That is why many companies have started offering Cloud-based Big Data solutions (like Microsoft HDInsight, Amazon Elastic MapReduce, or Google Cloud Dataproc), while IDC estimates that, by 2020, nearly 40% of Big Data analyses will be supported by public Clouds [9]. The advantages of this approach are manifold. For instance, it provides an effective and cheap solution for storing huge amounts of data, whereas

• *E. Gianniti, M. Ciavotta and D. Ardagna are with Politecnico di Milano.*

the pay-per-use business model allows to cut upfront expenses and reduce cluster management costs. Moreover, the elasticity can be exploited 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., in terms of nodes type and number) is very large, thus identifying the exact cluster configuration is a complex task; especially in the light of the consideration that the blend of job classes in a specific workload and their resource requirements may also vary over time.

At the very beginning, MapReduce jobs were meant to run on dedicated clusters to support batch analyses via a FIFO scheduler [10], [11]. 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 [12], [13]. This is not a loose requirement, indeed, as one of the major challenges [14], [15] is to predict the application execution times with sufficient degree of accuracy. 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 [15].

Our focus in this paper is to introduce D-SPACE4Cloud, a software tool designed to help system administrators and operators in the capacity planning of shared Big Data clusters hosted in the Cloud, so as to support both batch and interactive applications with deadline guarantees. We believe that being able to successfully address this problem at design time enables developers and operators to make informed decisions about the technology to use, while also allowing for the full exploitation of the potential offered by the Cloud infrastructure.

We formulate the capacity planning problem by means of a mathematical model, with the aim of minimizing the cost of Cloud resources. The problem considers multiple virtual machine (VM) types as candidates to support the execution of Big Data applications (a.k.a. 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 VM available is not the best choice from either the performance or performance/cost ratio perspective [12], [16]. Through a search space exploration, our approach seeks the optimal VM type and number of nodes considering also specific Cloud provider pricing models (namely, reserved, on demand, and spot instances [17]). 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 [18]. DIA execution times are estimated by relying on a gamut of models, including machine learning (ML) and simulation based on queueing networks (QNs), stochastic Petri nets (SPNs) [19], as well as an ad hoc simulator, dagSim [20], 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.

We validate the accuracy of our solutions on real systems by performing experiments based on the TPC-DS industry benchmark for business intelligence data warehouse applications. Microsoft Azure HDInsight, Amazon EC2, and the CINECA Italian supercomputing center have been considered as target deployments. Our 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 percentage error we can achieve is within 30% of the measurements in the worst case, with an average error around 12% for QNs and as low as 3% when using dagSim. On top of this, we show that optimally choosing 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.

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 we focus on the formulation of the optimization problem and on the design time exploration algorithm implemented within our tool. In Section 5 we evaluate our approach by considering first the accuracy that can be achieved by our simulation models and then the overall effectiveness of the optimization method. Finally, in Section 6 we compare our work with other proposals available in the literature and draw the conclusions in Section 7.

2 D-SPACE4Cloud Architecture

The tool we present and discuss in this paper, namely D-SPACE4Cloud, has been developed within the DICE (Developing Data-Intensive Cloud Applications with Iterative Quality Enhancement) H2020 European research project [21]. The project aims at filling gaps in model-driven engineering with regard to the development of DIAs in Cloud environments, embracing the DevOps [22] culture. DICE is committed to developing an integrated ecosystem of tools and methodologies intended to streamline the DIA development process 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 the (dynamic and static) characteristics of the data to be processed as well as their impact on the performance of the components of an application. In addition, the project develops an IDE capable of supporting the managers, developers, and operators in quality-related decisions. The IDE enforces the iterative design refinement approach through a toolchain of both design and run time tools. The former cover simulation, verification, and optimization of deployment, whereas the latter encompass deployment, testing, and feedback analysis of monitoring data.

D-SPACE4Cloud is the DIA deployment optimization tool integrated in 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 quality of service (QoS) requirements

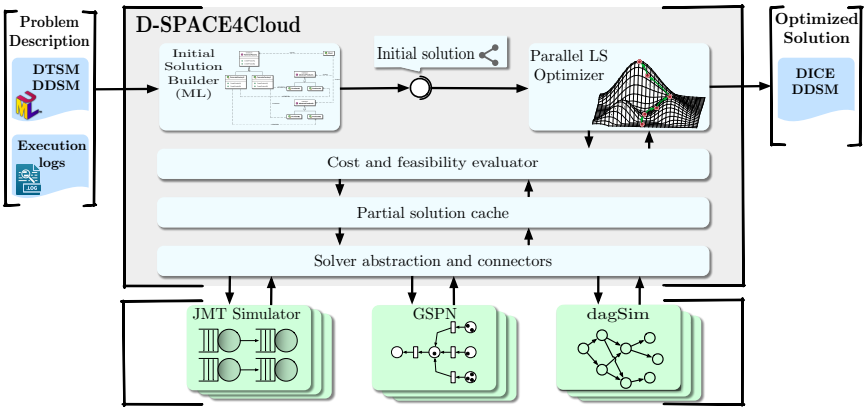


Figure 1. D-SPACE4Cloud architecture

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 the concurrent and on-time execution of several DIAs. To this end, the tool implements a design time exploration process able to consider multiple target VM candidates also across different Cloud providers. D-SPACE4Cloud supports the deployment optimization in the two distinct scenarios of public and a private Cloud environments. The public Cloud is mainly characterized by the fact that the cluster resources (i.e., VMs) can be considered practically infinite for any common purpose. In this scenario, concurrency level is not a problem and the tool focuses on selecting the most cost-effective VM type and number of replicas for each application. In the private Cloud scenario, however, the cluster is provisioned on premises, the available resources are generally limited, and the resource allocation plan has to contemplate the possibility to exhaust the computational capacity before being able to provision a cluster that satisfies the QoS constraints. In such a situation, the tool can, if required, alter the underlying problem considering a mechanism of admission control (i.e., including job rejection in the optimization process). In this paper, for space limitations, only the first scenario is presented, the discussion about motivations, algorithms, and models related to the second scenario is left to future publications.

Figure 1 depicts the main elements of D-SPACE4Cloud's 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 DICE IDE) devoted to handle the interactions with users and the other components of the toolchain, an optimization service (colored gray), which transforms the inputs into suitable performance models [19] 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 under the hood. Currently, D-SPACE4Cloud supports a QN simulator (JMT [23]), a SPN simulator (GreatSPN [24]), and discrete event simulator (dagSim [20]).

D-SPACE4Cloud takes in input:

1) a UML description of the applications sharing the cluster (see [22] for additional details on DICE UML models). In this context, DIAs are specified via *DICE Platform and Technology Specific Models* (DTSMs). Moreover, under specific circumstances, execution logs, for instance the ones obtained executing the applications in a pre-production environment, can replace the DTSMs as input.
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. This model is used as template to be filled and returned in output.
3) a description of the execution environment (list of candidate providers and VM types along with VM performance profiles). These pieces of information are used to generate suitable performance models.
4) the list of QoS constraints, that is the concurrency level and deadline for each DIA, respectively.

The optimization service is the centerpiece of the tool. 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, whose perhaps most interesting feature is that some of its constraints have been modeled by applying ML techniques to the problem of estimating the execution time of DIAs. Different techniques have been investigated [25], including linear regression, as well as Gaussian, polynomial, and linear support vector regression (SVR). The linear SVR was selected as it proved to be both accurate and robust to noisy data. More details are available in Section 4.

It must be highlighted, at this point, that the quality of the initial solution can still be improved, mainly because the MINLP relies on an approximate representation of the application-cluster liaison. For this reason, more accurate performance models (e.g., QNs and SPNs) are exploited. The increased accuracy creates room to maneuver for further cost reduction; however, since the simulation process is time-consuming, the space of possible cluster configurations has

to be explored in the most efficient way, avoiding the evaluation of unpromising configurations. The Optimizer component carries out this task, implementing a simulation-optimization technique to minimize the number of resource replicas (VMs) for each application class. This procedure is applied independently, and in parallel, on all the application classes and terminates when a further reduction in the number of replicas would lead to an infeasible solution.

As soon as all the classes reach convergence, D-SPACE4Cloud leverages the optimized solution (selected provider, type and number of VMs per application) to update the DDSM models and provides them in output. Such models, in turn, can be converted into TOSCA blueprints and used to deploy the cluster exploiting another tool, named DICER [22], part of the DICE toolchain.

3 Problem Statement

In this section we aim at introducing some important details on the problem addressed in this work. We envision the scenario wherein a business venture needs to set up a cluster to carry out efficiently a set of interactive DIAs. A cluster featuring the YARN Capacity Scheduler and running on a public Cloud infrastructure as a service (IaaS) is considered a fitting technological solution for the requirements of the company. In particular, the cluster has to support the parallel execution of DIAs in the form of Hadoop jobs and Hive/Pig/SparkSQL queries. Different classes $\mathcal{C} = \{i \,|\, i = 1, \ldots, n\}$ gather the applications that exhibit a similar behavior and share 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 application 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 data sets 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 speed, number of cores, available memory, etc.) and cost. Making the right design decision implies a remarkable endeavor that can be repaid 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, we consider a pricing model derived from Amazon EC2 [17]. 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 rate 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 data center capacity. For such instances customers bid and compete, yielding very competitive hourly fees at the expense of reduced guarantees on their reliability. In order to obtain the most cost-effective configuration, we rely on reserved VMs (denoting with r_i their number) to satisfy the core 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 costs 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 market fluctuations, to keep a high QoS level the number of spot VMs is bounded not to be greater than a fraction η_i of 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}, where every class may have a separate pool of reserved VMs of any type at their disposal. It is worth noting that this cost model is general enough to remain valid, zeroing the value of certain parameters, even in those cases where the considered Cloud does not feature on demand or spot instances.

In the remainder, we will denote by c_i 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.

Reducing the operating costs of the cluster by using efficiently the virtual resources in lease is in the interest of the company. This translates into a resource provisioning problem where the 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 [26].

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], [27], based on ML [16], [28], or the average execution times achieved via simulation: in this paper we adopted both the latter approaches.

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 this separation. Our 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, we can expect the performance estimate

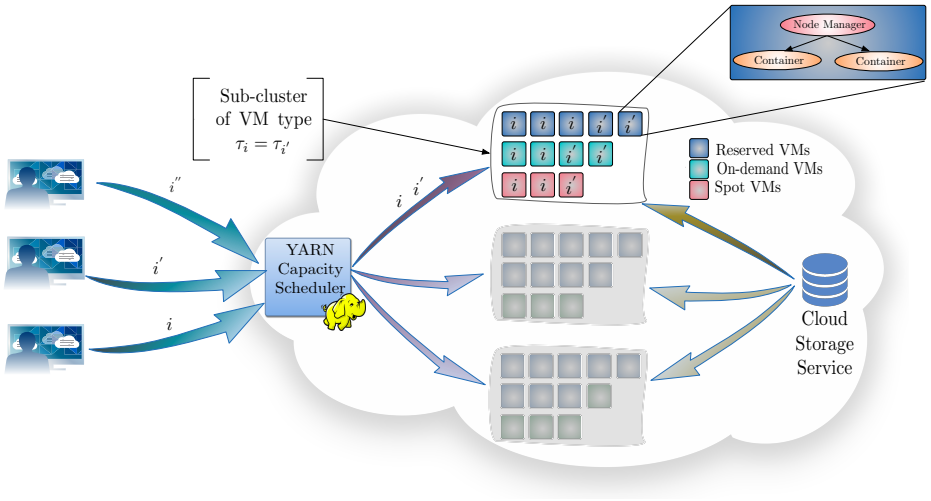


Figure 2. Reference system

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 cost for on demand VMs of type j [€/h]
σ_j	Unit hourly cost for spot VMs of type j [€/h]
R_{ij}	Number of reserved VMs of type j devoted to class i users

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. Equations (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, we can say that 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 cost is 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 problem. Furthermore, in 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.

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

4 Problem Formulation and Solution

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

The first issue D-SPACE4Cloud has to solve is to quickly (and 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, we decided to exploit ML models 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, yet 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 Sections 4.1 and 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 as 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:

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

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

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

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

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

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

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

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

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. We will enforce that only $x_{i,\tau_i} = 1$ in (P1b), 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:

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

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

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

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

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

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

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

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

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

In Problem (P2) we dropped ν_i exploiting constraints (P1k) and rewrote (P1l) as constraints (P2d)–(P2f). Specifically, (P2d) and (P2e) ensure that the overall number of containers, c_i, is consistent with nodes capacity, in terms of both vCPUs and memory. Constraint (P2f), on the other hand, is a simple 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.

Equation (P2f) 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. Building upon [25], which compares linear regression, Gaussian SVR, polynomial SVR with degree ranging between 2 and 6, and linear SVR, we follow [28] in opting for a model derived with linear SVR. This is due to the fact that SVR with other kinds of kernel fares worse than the linear one, whilst plain linear regression requires an ad hoc data cleaning to avoid linear dependencies in the design matrix, thus making it harder to apply in the greatest generality. In order to select a relevant feature set, we started by generalizing the analytical bounds for MapReduce clusters proposed in [15], [27]. This approach yielded 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 cores, of which we consider the reciprocal. 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., [30]).

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:

$$c_i \leq \alpha_i \left(r_i + d_i + s_i\right), \text{ where } \alpha_i \triangleq \min\left\{\frac{M_{\tau_i}}{m_i}, \frac{V_{\tau_i}}{v_i}\right\}. \tag{4}$$

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 ones will 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 Problem (P3) regularity, 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, we can obtain analytically 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:*

$$c_i = \frac{\chi^c_{i,\tau_i}}{D_i - \chi^h_{i,\tau_i} h_i - \chi^0_{i,\tau_i}}, \tag{5a}$$

$$\nu_i = \frac{c_i}{\alpha_i} = \frac{1}{\alpha_i} \frac{\chi^c_{i,\tau_i}}{D_i - \chi^h_{i,\tau_i} h_i - \chi^0_{i,\tau_i}}. \tag{5b}$$

Proof. Problem (P3) Lagrangian is given by:

$$\begin{aligned}\mathcal{L}\left(c_i, \nu_i\right) = \nu_i &+ \lambda_\alpha \left(c_i - \alpha_i \nu_i\right) + \\ &+ \lambda_\chi \left(\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} - D_i\right) + \\ &- \lambda_c c_i - \lambda_\nu \nu_i\end{aligned} \tag{6}$$

and stationarity conditions lead to:

$$\frac{\partial \mathcal{L}}{\partial \nu_i} = 1 - \alpha_i \lambda_\alpha - \lambda_\nu = 0, \tag{7a}$$

$$\frac{\partial \mathcal{L}}{\partial c_i} = \lambda_\alpha - \lambda_\chi \chi^c_{i,\tau_i} \frac{1}{c_i^2} - \lambda_c = 0, \tag{7b}$$

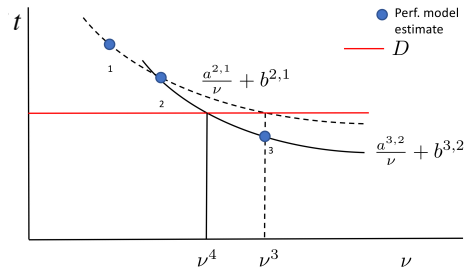


Figure 3. Hyperbolic jump

while complementary slackness conditions are:

$$\lambda_\alpha \left(c_i - \alpha_i \nu_i\right) = 0, \quad \lambda_\alpha \geq 0, \tag{8a}$$

$$\lambda_\chi \left(\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} - D_i\right) = 0, \quad \lambda_\chi \geq 0, \tag{8b}$$

$$\lambda_c c_i = 0, \quad \lambda_c \geq 0, \tag{8c}$$

$$\lambda_\nu \nu_i = 0, \quad \lambda_\nu \geq 0. \tag{8d}$$

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 (7a) and (7b) can be applied to obtain $\lambda_\alpha > 0$ and $\lambda_\chi > 0$. So constraints (P3c) and (P3d) are active in every optimal solution, whence we get (5a) and (5b). □

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 are executed in parallel as the original Problem (P1) has been split into independent sub-problems. After checking the feasibility of the initial solution, the search algorithm begins the exploration incrementing the VM count whenever the solution results infeasible or decreasing it to save on costs if current configuration is already feasible.

In order to avoid one-VM steps, which might lead to a very slow convergence for our 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], [27], [28]). Hence, at every iteration the application execution time is estimated as:

$$t_i = \frac{a_i}{\nu_i} + b_i, \tag{9}$$

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

Algorithm 1 Search algorithm

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

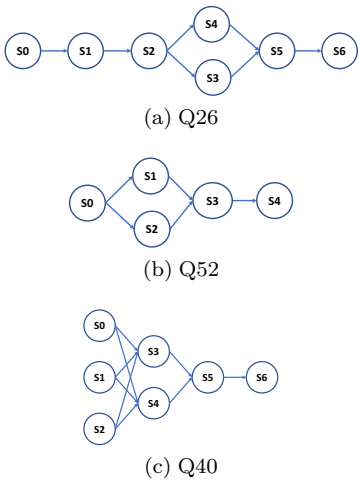
previous steps results. Hence, from the second search step on, we can compute a_i and b_i using the predicted execution times returned by the performance simulators and the associated resource allocations. In this way, at every iteration k it is possible to have an educated guess on the number of VMs required to meet the deadline D_i, as depicted in Figure 3:

$$\nu_i^{k+1} = \frac{a_i^{k,k-1}}{D_i - b_i^{k,k-1}}. \tag{10}$$

Our 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 (9) and (10). 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 (10) 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 searches iteratively the state space performing simulations and keeping track of the interval enclosing the optimal solution. Every new step relies on the hyperbolic function in (10), as shown at line 12.

As already 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 to speed 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. We address this issue 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, our 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

In this section we show the results of several experiments performed to validate the proposed approach. All these experiments have been performed on two Ubuntu 14.04 VMs hosted on an Intel Xeon E5530 2.40 GHz equipped server. 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, Section 5.2 validates the simulation models against the performance of real clusters, Section 5.3 presents a comparative study on the solutions obtained by varying the problem parameters identifying the potential savings of our approach. Section 5.4 is devoted to assess the quality of D-SPACE4Cloud solutions. Finally, the scalability of our approach is evaluated in Section 5.5.

5.1 Experimental Setup and Design of Experiments

To profile Big Data applications and compare with simulators results, we collected real measures by running SQL queries on Apache Hive[1] on MapReduce and Apache Spark[2]. We used the industry standard TPC-DS[3] benchmark data set generator to create synthetic data at scale factors ranging from 250 GB to 1 TB.

Figure 13, available in Appendix, shows 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 an example of MapReduce job. On the other hand, Q26, Q40, and Q52 are three of the queries that belong to the TPC-DS benchmark. These queries have been executed on SparkSQL, yielding the DAGs shown in Figure 4.

1. https://hive.apache.org
2. https://spark.apache.org
3. http://www.tpc.org/tpcds/

Since profiles collect statistical information about jobs, we repeated the profiling runs at least twenty times per query. Properly parsing the logs allows to extract all the parameters composing every query profile, for example 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[4], the Big Data cluster provided by CINECA, the Italian supercomputing center. The cluster created in EC2 was composed of 30 computational nodes, for a total of 120 vCPUs hosting 240 YARN containers, whilst on PICO we used up to 120 cores configured to host one container per core. On the other hand, on Azure clusters of variable sizes have been provisioned, reaching up to 26 dual-core containers. In the EC2 case every container had 2 GB RAM and on Cineca 6 GB, while on Azure containers were 2 GB for A3 machines, 8 GB for A4 and D12v2, 40 GB for D13v2, and 90 GB for D14v2. Along with profiles, we also collected lists of task execution times to feed the replayer in JMT service centers or dagSim stages. In the end, we recorded the different VM types characteristics.

Our previous work [19] 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. Our previous work also highlights that MapReduce and Spark stages tend to follow Erlang distributions, whose coefficient of variation is small.

5.2 Simulation Models Validation

To start off with, we show results for the validation of the different simulation models. We feed the models with parameters evaluated on the real systems we took into account and compare the measured performance metrics with the ones obtained via simulation.

Specifically, we consider as a quality index the relative error on the prediction of the execution times, defined as follows:

$$\vartheta = \frac{\tau - T}{T} \tag{11}$$

where τ is the simulated execution time, whilst T is the average measurement across multiple 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 possible deadline misses. Indeed, if $\vartheta < 0$ then the prediction is not conservative.

Among these experiments, we considered both single user scenarios, where one query has been run repeatedly on a dedicated cluster, interleaving a 10 s average think time between completions and subsequent submissions, and multiple user scenarios, with several users concurrently interacting with the cluster in a similar way.

The Appendix details the experimental campaign for simulation models validation. Table 7 shows the results of JMT QN models. In the worst case, the relative error can reach up to 30.59%, which is perfectly in line with the expected accuracy on execution time prediction for capacity planning purposes [26]. Instead, overall the average relative error settles at 12.27%. Moreover, Tables 5 and 6 report the results for dagSim models for single user scenarios on the 500 GB dataset. The worst case error is −19.01% and, on average, errors settle at 3.06%.

4. http://www.hpc.cineca.it/hardware/pico

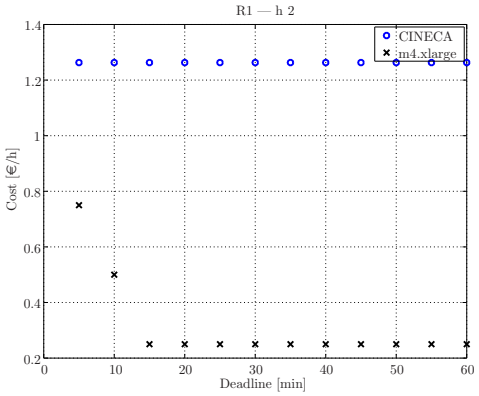


Figure 5. Query R1, two concurrent users

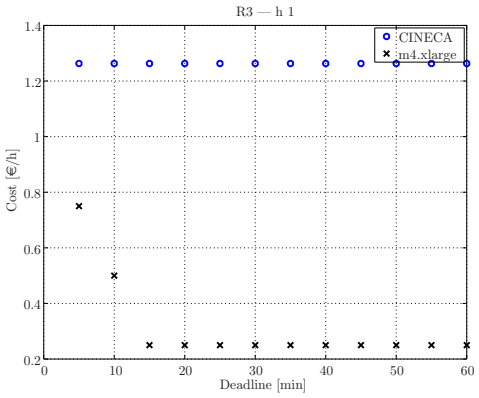


Figure 6. Query R3, one concurrent user

5.3 Scenario-based Experiments

The optimization approach described in Section 4 needs to be validated, ensuring that it is capable of catching realistic behaviors as one can reasonably expect of the system under analysis. We test this property with a set of assessment runs where we fix all the problem parameters but one and verify that the 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, we expect a need for more VMs to support the rising workload, thus leading to an increase of leasing costs. On the other hand, if at fixed parameters we tighten the deadlines D_i, again we should observe increased costs: the system will require a higher parallelism to shrink execution times, hence more containers to support it.

For the sake of clarity, in this section we performed single-class experiments: considering only one class ensures an easier interpretation of the results. Figures 5, 6 and 7 report the solutions obtained with the 250 GB data set on MapReduce queries when relying on the JMT simulator. The

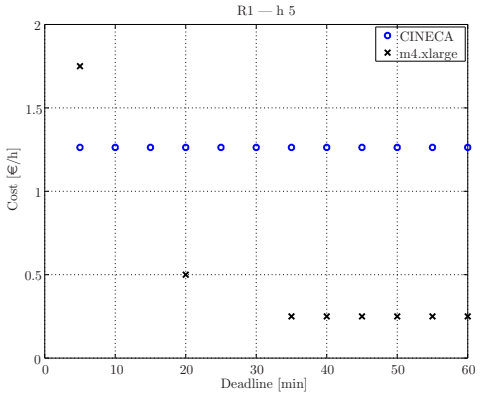


Figure 7. Query R1, five concurrent users

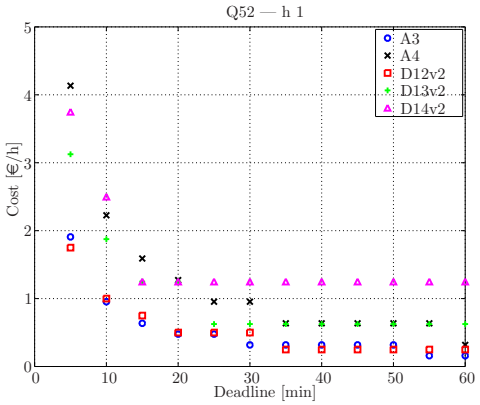


Figure 9. Query Q52, single user

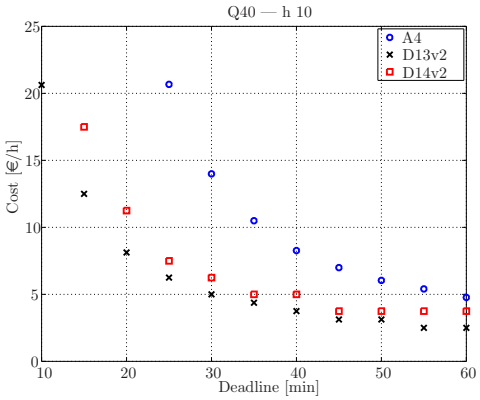


Figure 8. Query Q40, ten users

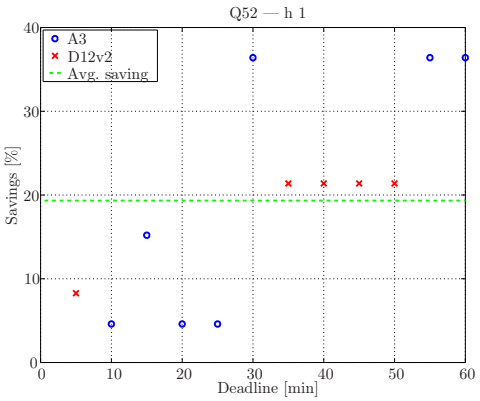


Figure 10. Query Q52, single user, savings

average running time for these experiments is about 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: CINECA is the 20-core node available on PICO, whilst m4.xlarge is the 4-core instance rented 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 Cloud cluster consisting of the smaller Amazon-offered instances, independently of the deadlines. It is then interesting to observe that R1 shows a different behavior if the required concurrency level increases. 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. Q40 with ten users exhibits a straightforward behavior, with D13v2 instances always leading to cheaper deployments. In order to quantify monetary savings, we compute the ratio of the difference between costs over the second cheapest alternative. 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. On the other hand, a single-user Q52 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. By identifying the proper alternative, it is possible to obtain an average saving around 19.3% over the considered range of deadlines. The maximum saving is about 36.4%.

Overall, these results provide a strong point in favor of the optimization procedure implemented in our tool, as they prove that making the right choice for deployment can lead to substantial savings throughout the application life cycle. Take into account that Microsoft Azure suggests VMs of the D11–15v2 range for memory intensive applications, such as analytics on Spark or distributed databases, whilst we showed that, depending on the workload, 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.

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

Finally, Figure 11 shows the results of a multi-user experiment over the 500 GB dataset. Fixing all the other parameters, in particular query Q26 and a deadline of 20 minutes, we varied the required concurrency level between 1 and 10 users with step 1. Here the D12v2 instances prove in every case the better choice, with an average 30.0% saving in comparison to

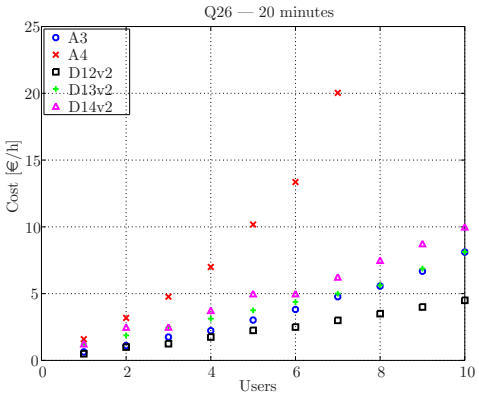


Figure 11. Query Q26, multi-user

Table 3
Optimizer single class validation, D12v2

Query	D [ms]	Cores	R [ms]	ε [%]
Q26	180,000	48	158,287	12.06
Q52	180,000	48	150,488	16.40
Q26	240,000	36	186,066	22.47
Q52	240,000	36	175,394	26.92
Q26	360,000	24	280,549	22.07
Q52	360,000	24	276,790	23.11

the second cheapest deployment.

5.4 Solution Validation in a Real Cluster Setting

A further experiment was aimed at assessing the quality of the optimized solution obtained using D-SPACE4Cloud. Given a query and a deadline to meet, we focus on the execution time measured in a real cluster provisioned according to the number and type of VMs returned by D-SPACE4Cloud, quantifying the relative gap as a metric of the optimizer accuracy. Formally:

$$\varepsilon = \frac{D - R}{D}, \tag{12}$$

where D is the deadline and R the execution time measured on the system, so that possible misses would yield a negative result.

We considered six cases, varying deadline and query, and ran D-SPACE4Cloud to determine the optimal resource assignment to meet the QoS constraints on D12v2 instances with a 500 GB scale factor. Table 3 collects data that relates to this experiment. Every row shows the relevant query and deadline, the optimal number of VMs, the measured execution time, and the percentage gap ε. First of all, none of the considered runs led to a deadline miss. Moreover, the relative error is always below 30%, with a worst case result of 26.92% and the average settling at 20.51%. Overall we can conclude that the optimization tool is effective in identifying the minimum cost solution at design time, guaranteeing that deadlines are met as well.

In addition, we used D-SPACE4Cloud to optimize a multi-class instance on D14v2 VMs. Table 4 shows the results of this experiment, with three different queries subject to the same deadline that share a cluster of three worker nodes, for

Table 4
Optimizer multi-class validation, D14v2

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

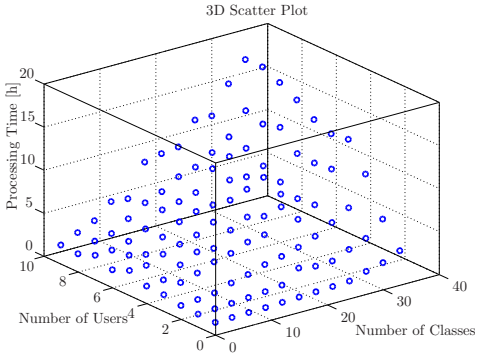


Figure 12. Execution time for varying number of classes and users

a total of 48 cores under a work conserving scheduling policy. In this case, the worst case result is 26.37%, with an average of 24.70%. The accuracy is slightly lower than the single class scenario due to possible performance gains allowed by spare resources borrowed from other classes during their idle time, yet it remains good for practical applications.

5.5 Scalability Analysis

In this section we quantify the time taken to obtain the optimized solution for instances of increasing size, both in terms of number of classes and aggregate user count. All these runs exploited 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, we have Q26, Q40, and Q52 with deadline 60 minutes, then the 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 3 classes has $D = 60$ minutes, the second instance with 6 classes collects $D \in \{60, 55\}$ minutes, the third one adds $D = 50$, and so forth. We repeated this test instance generation with a required level of concurrency ranging from 1 to 10 users, but without ever mixing classes with different h_i: in any given instance, $\forall i \in \mathcal{C},\ h_i = \bar{h}$. In this way, we considered a total of 120 different test instances 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 with 9 users each.

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 should be considered one order of magnitude larger than nowadays production environments.

6 Related Work

Capacity planning and architecture design space exploration are important problems analyzed in the literature [31], [32]. High level models and tools to support software architects (see, e.g., Palladio Component Model and its Palladio Bench and PerOpteryx design environment [33], [34], or stochastic process algebra and the PEPA Eclipse plugin [35], [36]) have been proposed for identifying the best configuration given a set of QoS requirements for enterprise Web-based systems, but unfortunately they do not support Cloud-specific abstractions or (see, e.g., [37]) directly address the problem of deriving an optimized Cloud and Big Data cluster configuration. On the other side, capacity management, cluster sizing, and tuning of Big Data applications have received also a widespread interest by both academia and industry.

The starting point of this second family of approaches is the consideration that 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 [38]. 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 to be exploited for Hadoop automatic configuration processes. Furthermore, also the cluster sizing problem has been tackled and successfully solved exploiting the same tool [39]. More recently, Dalibard et al. [40] have presented BOAT, a gray box framework, which supports developers to build efficient auto-tuners for their system, 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 computations.

The problem of job profiling and execution time estimation represents a common issue in the Big Data literature. Verma et al. [41] 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 [42]. 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. [43] 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.

The problem of progress estimation of multiple parallel queries is addressed in [44]. To this aim, the authors present Parallax, a tool able to predict the completion time of MapReduce jobs. The tool has been implemented over Pig, while the PigMix benchmark has been used for the evaluation. ParaTimer [45], an extension of Parallax, features support to multiple parallel queries expressed as DAGs.

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 [46]. Machine learning black box models are becoming also popular to predict the performance of large scale business analytics systems. Ernest [47] is a representative example of these approaches. The authors used experiment design to collect as few training points as required. As experimental analysis, the authors used Amazon EC2 and evaluated the accuracy of the proposed approach using several ML algorithms that are part of Spark MLlib. The evaluation showed that the average prediction error is under 20%, in line with our results.

In [28] we investigated a mixed analytical/ML approach to predict the performance of MapReduce clusters by relying on QNs to generate a knowledge base (KB) of synthetic data over which 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 we used for building the optimization models discussed in Section 4.

The capacity management and cluster sizing problems, instead, have been faced by Tian and Chen [48]. The goal is the minimization of the execution cost for a single MapReduce application. The authors present a cost model that depends on the dataset size and on some characteristics of the considered application. A regression-based analysis technique has been used to profile the application and to estimate model parameters.

MapReduce cluster sizing and 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 demonstrated the effectiveness of their approach comparing it with the offline generated optimal schedule.

Cluster sizing of MapReduce applications based on deadlines is considered in [49]. The authors recognize the inadequacy of Hadoop schedulers released at the date to properly handle completion time requirements. The work proposes to adapt to the problem some classical multiprocessor scheduling policies; 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 [50], where the authors present a solution to manage clusters shared among Hadoop application and more traditional Web systems.

Zhang et al. [50] investigate the performance of MapReduce applications on homogeneous and heterogeneous Hadoop clusters in the Cloud. They consider a problem similar to ours and provide a simulation-based framework for minimizing cluster infrastructural costs. Nonetheless, a single class of workload is optimized.

In [15] the ARIA framework is presented. This work is the closest to our contribution and 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. We borrow from this work the compact job profile definition, used there to calculate an estimation of an application execution time. The same authors, in a more recent work [12], provided 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.

Alipourfard et al. [51] have presented CherryPick, a black box system that leverages Bayesian optimization to find near-optimal Cloud configurations that minimize cloud usage cost for MapReduce and Spark applications. The authors' approach also guarantees application performance and limits the search overhead for recurring Big Data analytics jobs, focusing the search to improve prediction accuracy of those configuration close to the best for a specific deadline. With respect to this work, our approach allows for analyzing the trade-off between costs and performance and has demonstrated to provide good performance prediction and cost estimation across multiple configurations and deadlines.

Similar to our work, the authors in [52] provide a framework facing the problem of minimum cost provisioning of MySQL clusters in Cloud environments. The cost model includes resource costs and service level agreement penalties, which are proportional to execution time violation of a given deadline. Queries execution times are predicted through QN models, which, however, introduce up to 70% percentage error. The minimum cost configuration is identified through two greedy hill climbing heuristics, which can identify heterogeneous clusters, but no guarantees on the quality of the final solution can be provided. 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 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 demonstrated that their framework can manage effectively large systems, improving significantly infrastructure utilization and application performance. However, their collaborative filtering approach requires to gather little data from the running applications, but require 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.

7 Conclusions

In this paper we have proposed an effective tool for capacity planning of YARN managed Cloud clusters to support DIAs implemented as MapReduce jobs or through Apache Spark. We have developed a MINLP formulation based on ML models whose initial solution is iteratively improved by a sim-heuristic, which exploits simulation to assess accurately 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 proved how the tool is a valuable contribution towards supporting different application classes over heterogeneous resource types, since we have highlighted situations where choosing the best VM type is not trivial. Sometimes, 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.

Future work will extend D-SPACE4Cloud to support the resource provisioning of continuous applications that integrate batch and streaming workloads. Moreover, we will extend our solutions to the run time cluster management scenario.

References

[1] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big Data and its technical challenges," *Commun. ACM*, vol. 57, no. 7, pp. 86–94, Jul. 2014.

[2] (2017, Mar.) Worldwide semiannual Big Data and analytics spending guide. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=IDC_P33195

[3] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce: A survey," *SIGMOD Rec.*, vol. 40, no. 4, pp. 11–20, Jan. 2012.

[4] F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni, "Optimizing power and performance trade-offs of MapReduce job processing with heterogeneous multi-core processors," in *CLOUD Proc.*, 2014.

[5] C. Shanklin. Benchmarking Apache Hive 13 for enterprise Hadoop. [Online]. Available: https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

[6] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," *J. Parallel Distrib. Comput.*, vol. 74, no. 7, pp. 2561–2573, 2014.

[7] H. Derrick. (2015) Survey shows huge popularity spike for Apache Spark. [Online]. Available: http://fortune.com/2015/09/25/apache-spark-survey

[8] M. A. Greene and K. Sreekanti. (2016) Big Data in the enterprise: We need an "easy button" for hadoop. [Online]. Available: http://www.oreilly.com/pub/e/3643

[9] The digital universe in 2020. [Online]. Available: http://idcdocserv.com/1414

[10] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguadé, M. Steinder, and I. Whalley, "Performance-driven task co-scheduling for MapReduce environments," in *NOMS Proc.*, 2010.

[11] B. T. Rao and L. S. S. Reddy, "Survey on improved scheduling in Hadoop MapReduce in Cloud environments," *CoRR*, vol. abs/1207.0780, 2012. [Online]. Available: http://arxiv.org/abs/1207.0780

[12] 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.

[13] Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo, "Automated profiling and resource management of Pig programs for meeting service level objectives," in *ICAC Proc.*, 2012.

[14] M. Lin, L. Zhang, A. Wierman, and J. Tan, "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 *ICAC Proc.*, Jun. 2011.

[16] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna, "Stage aware performance modeling of DAG based in memory analytic platforms," in *IEEE CLOUD Proc.*, 2016.

[17] Amazon EC2 pricing. [Online]. Available: http://aws.amazon.com/ec2/pricing/

[18] Hadoop MapReduce next generation — Capacity Scheduler. [Online]. Available: http://hortonworks.com/blog/benchmarking-apache-hive-13-enterprise-hadoop/

[19] D. Ardagna, S. Bernardi, E. Gianniti, S. K. Aliabadi, D. Perez-Palacin, and J. I. Requeno, "Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets," in *ICA3PP Proc.*, 2016.

[20] A. B. et al. (2017) D3.4 EUBra-BIGSEA QoS infrastructure services intermediate version. [Online]. Available: http://www.eubra-bigsea.eu/sites/default/files/D3.4%20EUBra-BIGSEA%20QoS%20infrastructure%20services.pdf

[21] G. Casale and et al., "Dice: Quality-driven development of data-intensive cloud applications," in *MiSE Proc.*, 2015.

[22] M. Artac, T. Borovsak, E. D. Nitto, M. Guerriero, and D. A. Tamburri, "Model-driven continuous deployment for quality DevOps," in *QUDOS Proc.*, 2016.

[23] 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.

[24] Greatspn 7.2. [Online]. Available: http://www.di.unito.it/~greatspn/index.html

[25] A. M. Rizzi, "Support vector regression model for BigData systems," *ArXiv e-prints*, Dec. 2016.

[26] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance*. Prentice-Hall, 1984.

[27] M. Malekimajd, D. Ardagna, M. Ciavotta, A. M. Rizzi, and M. Passacantando, "Optimal Map Reduce job capacity allocation in Cloud systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 4, pp. 51–61, Jun. 2015.

[28] E. Ataie, E. Gianniti, D. Ardagna, and A. Movaghar, "A combined analytical modeling machine learning approach for performance prediction of MapReduce jobs in Cloud environment," in *SYNASC Proc.*, 2016.

[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] D. Ardagna, M. Ciavotta, and M. Passacantando, "Generalized nash equilibria for the service provisioning problem in multi-cloud systems," *IEEE Trans. Services Computing*, vol. 10, no. 3, pp. 381–395, 2017.

[31] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *Software Engineering, IEEE Trans. on*, vol. PP, no. 99, pp. 1–1, 2013.

[32] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev, "Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures," *Software Engineering, IEEE Trans. on*, vol. 41, no. 2, pp. 157–175, Feb. 2015.

[33] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3–22, 2009.

[34] A. Koziolek, H. Koziolek, and R. Reussner, "PerOpteryx: Automated application of tactics in multi-objective software architecture optimization," in *QoSA 2011 Proc.*, 2011.

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

[36] OMG. (2015) PEPA: Performance evaluation process algebra. [Online]. Available: http://www.dcs.ed.ac.uk/pepa/tools/

[37] J. Kross and H. Krcmar, "Model-based performance evaluation of batch and stream applications for Big Data," in *MASCOTS Proc.*, 2017.

[38] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for Big Data analytics," in *CIDR Proc.*, 2011.

[39] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *SOCC Proc.*, 2011.

[40] V. Dalibard, M. Schaarschmidt, and E. Yoneki, "BOAT: Building auto-tuners with structured Bayesian optimization," in *WWW Proc.*, 2017.

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

[42] S. Bardhan and D. A. Menascé, "Queuing network models to predict the completion time of the map phase of MapReduce jobs," in *Int. CMG Conference*, 2012.

[43] E. Vianna, G. Comarela, T. Pontes, J. M. Almeida, V. A. F. Almeida, K. Wilkinson, H. A. Kuno, and U. Dayal, "Analytical performance models for MapReduce workloads," *International Journal of Parallel Programming*, vol. 41, no. 4, pp. 495–525, 2013.

[44] K. Morton, M. Balazinska, and D. Grossman, "ParaTimer: A progress indicator for MapReduce DAGs," in *SIGMOD Proc.*, 2010.

[45] K. Morton, A. Friesen, M. Balazinska, and D. Grossman, "Estimating the progress of MapReduce pipelines," in *ICDE Proc.*, 2010.

[46] A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri, "Exploiting mean field analysis to model performances of Big Data architectures," *Future Generation Computer Systems*, vol. 37, no. 0, pp. 203–211, 2014.

[47] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *NSDI Proc.*, 2016.

[48] F. Tian and K. Chen, "Towards optimal resource provisioning for running MapReduce programs in public Clouds," in *CLOUD Proc.*, 2011.

[49] L. T. X. Phan, Z. Zhang, Q. Zheng, B. T. Loo, and I. Lee, "An empirical analysis of scheduling techniques for real-time Cloud-based data processing," in *SOCA Proc.*, 2011.

[50] W. Zhang, S. Rajasekaran, S. Duan, T. Wood, and M. Zhu, "Minimizing interference and maximizing progress for Hadoop virtual machines," *SIGMETRICS Perf. Eval. Review*, vol. 42, no. 4, pp. 62–71, 2015.

[51] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively unearthing the best Cloud configurations for big data analytics," in *NSDI Proc.*, 2017.

[52] R. Mian, P. Martin, and J. L. Vazquez-Poletti, "Provisioning data analytic workloads in a cloud," *Future Gener. Comput. Syst.*, vol. 29, no. 6, pp. 1452–1458, Aug. 2013.

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

[54] ——, "Quasar: Resource-efficient and QoS-aware cluster management," in *ASPLOS Proc.*, 2014.

Eugenio Gianniti is currently pursuing his Ph.D. degree in Computer Engineering 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 Roma Tre, Italy in 2008. From 2012 he is Postdoctoral Fellow at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano. His research work focuses on modeling and optimization of complex real-life problems mainly arising in the fields of scheduling and planning, and more recently resource management for Cloud based and data intensive systems under constraints of quality of service.

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.

Appendix

```
select avg(ws_quantity),
       avg(ws_ext_sales_price),
       avg(ws_ext_wholesale_cost),
       sum(ws_ext_wholesale_cost)
from web_sales
where (web_sales.ws_sales_price between 100.00 and
    150.00) or (web_sales.ws_net_profit
between 100 and 200)
group by ws_web_page_sk
limit 100;
```

(a) R1

```
select avg(ss_quantity), avg(ss_net_profit)
from store_sales
where ss_quantity > 10 and ss_net_profit > 0
group by ss_store_sk
having avg(ss_quantity) > 20
limit 100;
```

(b) R3

```
select i_item_id,
avg(cs_quantity) agg1,
avg(cs_list_price) agg2,
avg(cs_coupon_amt) agg3,
avg(cs_sales_price) agg4
from catalog_sales, customer_demographics, date_dim,
    item, promotion
where catalog_sales.cs_sold_date_sk = date_dim.
    d_date_sk
and catalog_sales.cs_item_sk = item.i_item_sk
and catalog_sales.cs_bill_cdemo_sk =
    customer_demographics.cd_demo_sk
and catalog_sales.cs_promo_sk = promotion.p_promo_sk
and cd_gender = 'F'
and cd_marital_status = 'W'
and cd_education_status = 'Primary'
and (p_channel_email = 'N' or p_channel_event = 'N')
and d_year = 1998
group by i_item_id
order by i_item_id
limit 100;
```

(c) Q26

```
select dt.d_year, item.i_brand_id brand_id, item.
    i_brand brand, sum(ss_ext_sales_price) ext_price
from date_dim dt, store_sales, item
where dt.d_date_sk = store_sales.ss_sold_date_sk
and store_sales.ss_item_sk = item.i_item_sk
and item.i_manager_id = 1
and dt.d_moy=12
and dt.d_year=1998
group by dt.d_year, item.i_brand, item.i_brand_id
order by dt.d_year, ext_price desc, brand_id
limit 100;
```

(d) Q52

Figure 13. Queries

Table 5
dagSim model validation, Microsoft Azure D12v2

Query	Cores	Tasks	T [ms]	τ [ms]	ϑ [%]
Q26	12	1,406	660,700	620,773	−6.04
Q52	12	704	658,397	654,464	−0.60
Q26	16	1,406	551,669	495,246	−10.23
Q52	16	704	515,202	512,122	−0.60
Q26	20	1,406	454,054	393,414	−13.36
Q52	20	704	410,588	407,066	−0.86
Q26	24	1,406	385,639	332,364	−13.81
Q52	24	704	356,296	353,852	−0.69
Q26	28	1,406	354,183	286,861	−19.01
Q52	28	704	302,741	299,305	−1.13
Q26	32	1,406	304,048	250,327	−17.67
Q52	32	704	263,034	260,648	−0.91
Q26	36	1,406	244,214	228,456	−6.45
Q52	36	704	245,084	242,489	−1.06
Q26	40	1,406	225,484	208,327	−7.61
Q52	40	704	213,353	211,291	−0.97
Q26	44	1,406	198,966	189,840	−4.59
Q52	44	704	198,044	196,234	−0.91
Q26	48	1,406	186,659	186,953	0.16
Q52	48	704	188,860	187,162	−0.90
Q26	52	1,406	170,516	171,346	0.49
Q52	52	704	177,380	175,511	−1.05

Table 6
dagSim model validation, Microsoft Azure A3

Query	Cores	Tasks	T [ms]	τ [ms]	ϑ [%]
Q26	6	1,406	2,475,150	2,479,524.66	0.18
Q52	6	704	2,101,121	2,094,742.67	−0.30
Q26	8	1,406	2,014,112	2,026,360.58	0.61
Q52	8	704	1,651,055	1,644,624.56	−0.39
Q26	10	1,406	1,718,490	1,720,192.80	0.10
Q52	10	704	1,270,516	1,258,821.03	−0.92
Q26	12	1,406	1,632,222	1,647,299.29	0.92
Q52	12	704	1,067,327	1,059,946.54	−0.69
Q26	14	1,406	1,381,072	1,393,737.17	0.92
Q52	14	704	918,809	913,134.69	−0.62
Q26	16	1,406	1,213,972	1,224,156.94	0.84
Q52	16	704	827,597	823,043.16	−0.55
Q26	18	1,406	1,069,438	1,095,197.38	2.41
Q52	18	704	759,571	752,902.94	−0.88
Q26	20	1,406	1,036,132	1,035,922.42	−0.02
Q52	20	704	681,948	676,836.53	−0.75
Q26	22	1,406	919,943	989,514.49	7.56
Q52	22	704	608,599	603,718.41	−0.80
Q26	24	1,406	850,542	872,744.23	2.61
Q52	24	704	561,149	556,509.52	−0.83
Q26	26	1,406	657,342	671,679.98	2.18
Q52	26	704	507,889	504,324.45	−0.70
Q52	28	704	474,160	470,658.54	−0.74
Q26	30	1,406	586,840	625,812.40	6.64
Q26	32	1,406	565,578	579,209.02	2.41
Q26	34	1,406	561,356	583,397.80	3.93
Q52	34	704	397,761	392,896.90	−1.22
Q26	36	1,406	511,154	536,921.19	5.04
Q52	36	704	377,816	374,978.62	−0.75
Q26	38	1,406	482,202	507,705.68	5.29
Q52	38	704	375,542	373,554.58	−0.53
Q26	40	1,406	466,190	491,614.12	5.45
Q52	40	704	354,247	351,353.37	−0.82
Q26	42	1,406	425,101	447,379.84	5.24
Q52	42	704	329,417	327,510.50	−0.58
Q26	44	1,406	406,187	429,318.34	5.69
Q52	44	704	321,173	316,978.11	−1.31
Q26	46	1,406	383,123	391,511.05	2.19
Q52	46	704	314,163	316,043.45	0.60
Q26	48	1,406	367,084	398,411.46	8.53
Q52	48	704	300,379	296,947.31	−1.14

Table 7
JMT QN model validation, Amazon and Cineca

Query	h_i	Cores	Dataset [GB]	Map tasks	Reduce tasks	T [ms]	τ [ms]	ϑ [%]	Provider
R1	1	240	250	500	1	55,410	50,753	−8.40	Amazon
R3	1	240	250	750	1	76,806	77,260	0.60	Amazon
R1	1	60	500	287	300	378,127	411,940	8.94	Cineca
R3	1	100	500	757	793	401,827	524,759	30.59	Cineca
R3	1	120	750	1,148	1,009	661,214	759,230	14	Cineca
R3	1	80	1,000	1,560	1,009	1,019,973	1,053,829	−1.00	Cineca
R1	3	20	250	144	151	1,002,160	1,038,951	3.67	Cineca
R1	5	20	250	144	151	1,736,949	1,215,490	−30.02	Cineca
R1	5	40	250	144	151	636,694	660,241	3.70	Cineca