Optimal Resource Allocation of Cloud-Based Spark Applications

Danilo Ardagna, Enrico Barbierato, Eugenio Gianniti, and Marco Lattuada

Abstract—Nowadays, the big data paradigm is consolidating its central position in the industry, as well as in society at large. Lots of applications, across disparate domains, operate on huge amounts of data and offer great advantages both for business and research. According to analysts, cloud computing adoption is steadily increasing to support big data analyses and Spark will probably take a prominent market position for the next decade.
As big data applications gain more and more importance over time and given the dynamic nature of cloud resources, it is fundamental to develop intelligent resource management systems to provide Quality of Service guarantees to application end-users.
This paper presents a set of run-time optimization-based resource management policies for advanced big data analytics. Users submit Spark applications characterized by a priority, and by a hard or soft deadline. Optimization policies identify the minimum capacity to run a Spark application within the deadline and re-balance the cloud resources in case of heavy load, minimising the weighted applications tardiness. The solution relies on an initial non-linear programming model formulation and a search space exploration based on simulation-optimization procedures. Spark applications execution times are estimated by relying on a gamut of techniques, including machine learning, approximated analyses, and simulation. The benefits of the approach are evaluated on Microsoft Azure HDInsight and on a private cloud cluster based on Power 8 by considering the TPC-DS industry benchmark. The results demonstrate that the percentage error of the prediction of the optimal resource usage with respect to system measurement is around 8% on average. Moreover, the proposed algorithms can address complex problems like computing the optimal redistribution of resources among tens of applications in few minutes.

Index Terms—Big Data; Quality of Service; Elastic resource provisioning; Cluster management.

1 Introduction

MANY analysts point out that in recent years 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 reports that big data used to concern highly experimental projects, but now they are used in production [2]. The market is growing from $130 billion in 2016 to $203 billion in 2020, with a compound annual growth rate of 11.9 %, with the banking and manufacturing industries leading in terms of investments [3]. Moreover, data mining along with big data analytics are heavily changing our society [4], e.g., in the financial sector [5] or in healthcare [6].

Cloud computing is an enabling cost-effective technology for big data. It provides an *easy button* that allows users to automatically setup pre-configured clusters. Many companies have started offering cloud-based big data solutions, such as Microsoft HDInsight, Amazon Elastic MapReduce, or Google Cloud Dataproc; IDC estimates that, by 2020, nearly 40 % of big data analyses will be supported by public clouds [7]. On the economic side, cloud deployments are a convenient solution for storing huge amounts of data, whereas the pay per use business model and the elasticity allow cutting upfront expenses and reduce cluster management costs.

Given the central role of big data in our society and the dynamic nature of both cloud environments and big data frameworks, it is of paramount importance today to develop intelligent resource management systems, which provide Quality of Service (QoS) guarantees to application end-users while providing an efficient use of resources [8]. On one hand, big data cloud infrastructures include high-end I/O and memory optimized servers and support multiple competing applications where resource contention might lead to performance decline. On the other hand, in modern frameworks like Apache Hadoop 3 and Spark, resources are dynamically allocated among ready tasks. This allows a better cluster utilization, but providing deadline guarantees to big data analyses has become much more difficult.

In this paper, run-time optimization based resource management policies for advanced big data analytics are proposed. Specifically, they can efficiently manage cloud resources providing differentiated service levels to Spark end-users. Indeed, each user can submit jobs to be executed characterized by a priority and by a deadline, which can be hard or soft. The proposed solution identifies the minimum capacity to run a Spark application within an a priori deadline and rebalance the resources of the applications in case of heavy load, minimising the weighted application tardiness.

The run-time resource management problem is formulated by means of mathematical models, with the aim of optimizing the usage of the infrastructural resource. Through a search space exploration, our approach seeks the optimal virtual machines (VMs) number for each application. The underlying optimization problems are tackled by simulation-optimization procedures able to determine the best configuration. Spark applications execution times are estimated by relying on a gamut of techniques, including machine learning (ML), ap-

• *D. Ardagna, E. Barbierato, E. Gianniti and M. Lattuada are with Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Via Golgi 42, 20133 Milano, Italy. E-mail: {name.lastname}@polimi.it*

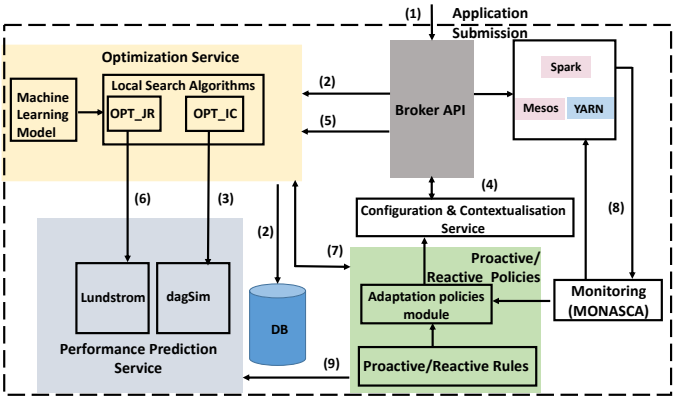


Figure 1. EUBra-BIGSEA run-time framework

proximated analyses, and simulation [9].

The accuracy of the proposed solutions is evaluated on Microsoft Azure HDInsight and on a private cluster based on Power 8 by performing experiments based on the TPC-DS industry benchmark for business intelligence data warehouse applications. The presented approach proved to achieve good performance: The experiments performed on real systems have shown that the percentage error is within 32% of the measurements in the very worst case, with an average error around 8%. An extensive scalability analysis demonstrates that the proposed algorithms can address complex problems like computing the optimal redistribution of shared resources among tens of running applications in about three minutes.

The proposed solutions have been developed within the EUBra-BIGSEA project [1] fostering a general framework to automate the deployment and the resource management of big data applications with QoS guarantees which is described in the next section. Section 3 introduces the problem formulation and the proposed solution to identify the capacity needed to support the execution of a Spark application within a deadline. Section 4 is devoted to the resource re-balancing problem, while Section 5 discusses the experimental results that validate the proposed solutions. Section 6 overviews other literature proposals. Conclusions are finally drawn in Section 7.

2 EUBra-BIGSEA Runtime Architecture

The EUBra-BIGSEA runtime framework consists of a set of key components designed to satisfy a group of requirements, specifically: i) handling the resource provision, ii) reducing costs related to big data application execution, and iii) guaranteeing QoS. All the modules are driven by the optimization of the infrastructure resources usage, which is accomplished by monitoring and dynamically allocating resources to meet deadlines.

The considered Spark applications can be executed on Mesos or, alternatively, on YARN resource managers. Given an application, its deadline can be classified as *hard* or *soft*. Hard deadlines must be fulfilled; soft deadlines have a priority associated and can be violated if the system does not have enough capacity. In this case, the system tries to minimize the weighted tardiness of the soft deadline applications (i.e., it will possibly reallocate the cluster nodes in a way that the weighted sum of application exceeding times with respect to soft deadlines is minimized).

The applications running on the EUBra-BIGSEA platform are profiled in advance and their resource allocation is obtained through optimization-based policies on the basis of the performance requirements. Applications are profiled according to a set of metrics including: i) the number of stages, ii) the number of tasks per stage, iii) their average execution time, and iv) the stage dependencies. Note that the lasts are modelled as a directed acyclic graph (DAG).

As shown in Figure 1, the architecture consists of a *Broker*, a *Configuration and Contextualization Service*, a *Monitoring* system, a *Performance Prediction Service* and two software management layers, i.e., the *Optimization Service* and the *Proactive/Reactive Policies* module.

The entry point of the system is the *Broker*, which receives application submissions enriched with additional information, such as the configuration (in terms of VM type, executors memory, executors cores, etc.) and the deadlines, and it triggers the application execution.

The optimal configuration for each application is computed by the *Optimization Service*, which aims at pursuing the respect of QoS guarantees and reducing the resource usage costs. It is composed of two sub-components: i) *OPTimizer Initial Configuration (OPT_IC)*, a module to provide the initial capacity configuration (described in Section 3) and ii) *OPTimizer Job Rebalancer (OPT_JR)*, a module to rebalance the applications capacities in case of heavy load (described in Section 4).

To compute the optimal solutions, the *Performance Prediction Service* is exploited: it is composed of two predictor tools and estimates the application execution time given the total number of available cores. Specifically, OPT_IC exploits an ad-hoc discrete event simulator called *dagSim* [9] providing off-line accurate results at the cost of long execution times (in the order of seconds/minutes). On the other end, OPT_JR uses a different predictor called *Lundstrom* [9] that provides on-line less accurate results though performing well in terms of execution time (sub-second) even for large DAGs.

Reactive and Proactive Policies module provides a bridge between the users' application submission and the execution platform, by adding (or removing) resources according to the threshold reactions triggered by the monitoring infrastructure. Violations (or even over-provisioning) trigger the execution of high-level rules resulting in the adaptation of the infrastructure by horizontally or vertically scaling the resources currently deployed and balancing the load among the physical servers. The control of the infrastructure is demanded to this module by exploiting the performance prediction service in order to understand if an application is early or late with respect to the deadline (an example of a rule is the following: if the delay is greater than 10%, then add 20% more nodes). The actual implementation of the solutions is demanded to the *Configuration & Contextualisation Service*, which builds the initial deployment and applies the horizontal scaling as suggested by the *Reactive and Proactive Policies* module.

Finally, the *Monitoring* system fulfils the task of collecting information regarding the system metrics, such as batch queues capacity, network and CPU load, or applications elapsed time.

1. http://www.eubra-bigsea.eu

Since optimization-based policies are effective only when application performance profiling is performed, the developed techniques consider mainly batch applications (such as data acquisition operations, bus trajectory identification, and social data clustering) while interactive applications are managed through reactive and proactive rule-based policies. Rules of this kind, specified by system administrators, are based on the analysis of the monitoring metrics and trigger system reconfiguration if the monitoring metric (reactive rule) or a prediction of the monitoring metric (pro-active rule) is above or below a given threshold.

As per Figure 1, the infrastructure workflow can be described according to the system load conditions. When an application is submitted to the *Broker* by specifying an application description (i.e., an identifier, the associated deadline, and other parameters), OPT_IC determines the number of VMs to be assigned to application execution. Then two cases can arise: either the system has enough capacity to accommodate the execution of the submitted application (light load) or this can start only by borrowing resources (heavy load) from soft deadline applications. In particular:

- *Light load*: Initially, an application with a deadline is submitted to the system (1), the optimization-based module verifies (2) if the application was already executed. Either the optimal configuration (that describes the number of VMs to run and their configuration in terms of RAM and number of cores) is found in the system history or the optimal configuration is obtained by interpolating the configurations available in the history (e.g., considering previous configurations corresponding to different deadlines) and by calculating the real value later (3). The *Broker* verifies that the system has enough capacity for the application to be executed; the actual execution starts after the application resource pool is allocated by the *Configuration and Contextualisation Service* (4).
- *Heavy load*: In this scenario, after steps (1) and (2), the broker verifies that the system has not enough capacity for the application execution and triggers a reconfiguration operation to the optimization-based module (5), which exploits Lundstrom (6) to determine the new number of VMs for all the soft deadline applications. The *Optimization Service* provides the new application configuration to the proactive policies module (7), which actuates the changes in the system.

In both the two load conditions, the *Proactive/Reactive Policies* module periodically checks the status of applications execution (8) and polls the *Performance Prediction Service* (9) to predict, given the current system configuration, the application finishing time.

If a deadline violation is predicted or the resource pool is over-provisioned, the proactive policies module adapts the system configuration according to its rules (e.g., increasing CPU frequencies for the underlying nodes, increasing the RAM devoted to execution, starting or stopping VMs, etc.), possibly triggering the execution of the optimization service.

In the next section, the initial capacity allocation problem is formulated; the problem under heavy load conditions is described instead in Section 4.

3 Identifying the Initial Configuration

The aim of the OPT_IC component is to identify the minimum number of VMs to be allocated to a new application submitted to the system to fulfill its deadline. The service, jointly with the monitoring system, can also periodically estimate the capacity needed by a running application to complete within its deadline according to its progress. In the beginning, for the sake of simplicity, only the former scenario is considered.

OPT_IC computes the application resource need by relying on a mixed integer nonlinear programming (MINLP) formulation and improves it via a simulation-optimization algorithm. It must be highlighted, at this point, that the quality of the initial solution can still be improved, mainly because some of the constraints of MINLP rely on an approximate performance model obtained via ML. If the strong suit of these techniques is the regression within the range explored during the training phase, there is no guarantee on their accuracy when generalized to other regions of the state space. Since the profiling activity is time consuming, it is quite unlikely that extensive information is available for the initial training of the ML model, hence simulation-based performance models are exploited, which achieve a good accuracy without depending too much on the particular configuration under investigation. The difference in accuracy might cause the need to adjust the solution with a local search; however, since even the simulation process is time consuming, the space of possible cluster configurations has to be explored in the most efficient way, avoiding the simulation of unpromising configurations. OPT_IC carries out this task, implementing a simulation-optimization technique to minimize the number of resources (i.e., VMs) for each application. This procedure terminates when a further reduction in the number of resources would lead to an infeasible solution (i.e., deadline violation). The optimized solution (number of VMs to be provided to the application) is then returned to the *Broker* and deployed through the *Configuration and Contextualization Service*.

In the following, some important details on the capacity allocation problems addressed in this paper are introduced. $\mathcal{A} = \{i \,|\, i = 1, \ldots, n\}$ denotes the set of running/submitted applications, with $\mathcal{A}^{\mathrm{d}} \subseteq \mathcal{A}$ the set of soft deadline applications, each of which is characterized by a weight w_i that formalizes its (or its end user's) priority: the higher the weight, the more important is to meet the prearranged deadline D_i. Each application is assumed to be executed by only one preassigned VM type characterized by Γ_i cores per VM. ν_i denotes the total number of allocated VMs, with $c_i = \Gamma_i \nu_i$ the total number of cores devoted to application i, and with N the total number of cores available for soft deadline applications.

Moreover, it is assumed that YARN and Mesos are 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 order to rigorously model and solve the capacity allocation problem, it is crucial to predict with fair confidence the execution times of each application under different cluster sizes. The execution time can be expressed as a function of the number of allocated cores. In general, any such performance

Table 1
Model parameters

Param.	Definition
$\mathcal{A}$	Set of applications
$\mathcal{A}^{\mathrm{d}} \subseteq \mathcal{A}$	Set of soft deadline applications
D_i	Deadline associated to application i [s]
w_i	Weight associated to application i
Γ_i	Number of CPUs available within a VM of application i
χ_i^c	Coefficient associated to $\frac{1}{c_i}$ in the model for application i [s]
χ_i^0	Constant term in the model for application i [s]

Table 2
Decision variables

Var.	Definition
ν_i	Number of VMs devoted to application i execution
c_i	Total number of cores assigned to application i

model needs to be based on a preliminary profiling of the application in order to be representative of its performance. In particular, in [10] it is demonstrated that big data applications execution time can be approximated, via support vector regression, by:

$$\mathcal{T}(c_i) = \chi_i^c \frac{1}{c_i} + \chi_i^0. \tag{1}$$

Equation (1) is the result of a ML process to get a first order approximation of the execution time of Hadoop or Spark jobs in cloud clusters. In order to select a relevant feature set, the approach for the analytical bounds for MapReduce clusters proposed in [11], [12] is generalized. 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 the reciprocal is considered. Since most of these features characterize the application class but cannot be controlled, equation (1) collapses all but c_i, with the corresponding coefficients, into a single constant term, χ_i^0, that is the linear combination of the feature values with the SVR-derived weights.

Table 1 reports a complete list of the parameters used in the models presented in this and in the next section, whilst Table 2 summarizes the decision variables.

The overall goal of OPT_IC is to determine the minimum number of VMs ν_i and cores c_i, such that the submitted application i is executed within its deadline D_i:

$$\min_{c_i,\nu_i} \nu_i \tag{P1a}$$

subject to:

$$c_i = \Gamma_i \nu_i, \tag{P1b}$$

$$\chi_i^c \frac{1}{c_i} + \chi_i^0 \leq D_i, \tag{P1c}$$

$$c_i \in \mathbb{N}, \tag{P1d}$$

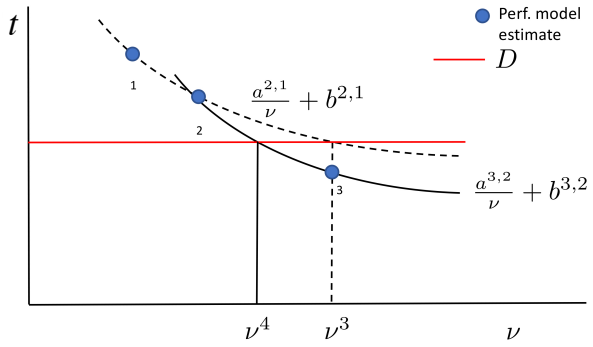


Figure 2. Hyperbolic jump

$$\nu_i \in \mathbb{N}. \tag{P1e}$$

The objective (P1a) is to minimize the number of VMs needed for the submitted application. Constraint (P1b) ensures that the overall number of cores is consistent with the VMs capacity. Further, constraint (P1c) exploits Equation 1 to enforce that the completion time meets the arranged deadline D_i. In the end, (P1d)–(P1e) provide the definition domains of the variables.

The nonlinearity in Problem (P1) can be easily tackled via an algebraic transformation of constraint (P1c). In this way, the optimum can be analytically obtained in closed form, as proven in Theorem 1.

Theorem 1. *The optimal solution of Problem* (P1) *is:*

$$c_i = \left\lceil \frac{\chi_i^c}{D_i - \chi_i^0} \right\rceil, \tag{2a}$$

$$\nu_i = \left\lceil \frac{c_i}{\Gamma_i} \right\rceil = \left\lceil \frac{1}{\Gamma_i} \left\lceil \frac{\chi_i^c}{D_i - \chi_i^0} \right\rceil \right\rceil. \tag{2b}$$

Proof. Via algebraic transformations, constraint (P1c) becomes:

$$c_i \geq \frac{\chi_i^c}{D_i - \chi_i^0}, \tag{3}$$

whence the minimum is obtained at the next greater integer. Similarly for ν_i. □

The efficient space exploration of possible configurations by OPT_IC is described in Algorithm 1. ν_i^k identifies the current solution at iteration k, l_i^k (i.e., the most resource hungry infeasible solution) and u_i^k (i.e., the feasible configuration with fewest VMs) the current lower and upper bounds of the solution. OPT_IC starts from the initial solution ν_i^0 obtained via equations (2), but since (P1c) might be only a preliminary approximation, the very first step of the procedure is simulating the initial configuration in order to refine the prediction (line 1).

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 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

Algorithm 1 Local search

Require: $\nu_i^0 \in \mathbb{N}$
1: $t_i^0 \leftarrow$ 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: $\quad t_i^k \leftarrow$ simulate ν_i^k
11: $\quad$ update l_i^k, u_i^k
12: $\quad \nu_i^{k+1} \leftarrow f\left(\nu_i^k, \nu_i^{k-1}, t_i^k, t_i^{k-1}\right)$
13: $\quad$ check ν_i^{k+1} against (l_i^k, u_i^k)
14: **until** $u_i^k - l_i^k = 1$
15: **return** u_i^k

exploits the fact that the execution time of a Spark application is inversely proportional to the allocated resources. Hence, at every iteration k the application execution time is estimated as:

$$t_i^{k+1} = \frac{a_i^{k,k-1}}{\nu_i^k} + b_i^{k,k-1}, \tag{4}$$

where t_i is the estimated execution time and ν_i the number of VMs, whilst a_i and b_i are obtained by fitting the hyperbola to the two previous steps results (see Figure 2). Hence, from the second search step on, a_i and b_i can be computed using the predicted execution times returned by the dagSim simulator and the associated resource allocations in the current iteration k and in the previous iteration $k-1$. 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 2, where hyperbolas obtained at subsequent steps are used to determine the next resource allocation to assess by intersection with the line at height D_i. The result of this operation is:

$$\nu_i^{k+1} = \frac{a_i^{k,k-1}}{D_i - b_i^{k,k-1}}. \tag{5}$$

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 (4) and (5). Since equation (5) 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 (5), as shown at line 12.

As already mentioned, OPT_IC mixes dichotomic search and domain knowledge about performance characteristics in order to exploit the best of both the worlds. Fitting a hyperbola to previous results allows the algorithm 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. This issue is addressed by updating at every iteration the upper bound u_i^k if last solution is feasible, or the lower bound l_i^k otherwise. 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 Problem (P1).

4 Resource Rebalancing under Heavy Load

At submission time, if the capacity available in the system is enough to accommodate the deployment of a new application, then its execution can start. When this is not the case (heavy load), the infrastructure will try to achieve the best possible QoS by prioritizing the allocation of resources to hard deadline applications and reallocating the residual capacity among soft deadline ones. Given the soft deadline applications execution progress, OPT_JR determines the new configuration that minimizes the weighted tardiness of soft deadline applications.

Note that, under heavy load conditions, the newly submitted application can start its execution only borrowing resources from soft deadline applications, hence it will receive the amount of resources established by OPT_IC if and only if: i) it is a hard deadline application and ii) the infrastructure has enough capacity to accommodate the requirements of all the hard deadline applications. Specifically, if the latter condition cannot be satisfied, the submission is rejected and system administrators are alerted of the failure.

Since this problem is solved while applications run, the monitoring system is accessed to assess whether applications are either executing faster than expected or falling behind their schedule. For this reason, provided that application i is executing stage s, a new rescaled deadline is computed as follows:

$$D_i' = \frac{t_i^{\mathrm{r}}}{t_i^{\mathrm{e}}} D_i, \tag{6}$$

where t_i^{r} is the elapsed time from the submission of the application obtained through the monitoring system, while t_i^{e} is the estimated time to execute the application up to the end of stage s obtained by dagSim. If the application is faster than predicted (i.e., it's early), the ratio is larger than 1, so the deadline is relaxed. On the contrary, if the application is slower than predicted (i.e., it's late), the ratio is smaller than 1, so the deadline is tightened. The same approach is exploited in OPT_IC to update the estimation of the number of necessary VMs during the application execution.

Given the ML formulation for application execution time, it is possible to define the tardiness of every application as:

$$\left(\chi_i^c \frac{1}{c_i} + \chi_i^0 - D_i'\right)^+, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{7}$$

This equation uses the positive part because the tardiness is 0 when an application is estimated to meet the deadline. The minimum weighted tardiness problem can then be formulated as follows:

$$\min_{\mathbf{c},\boldsymbol{\nu}} \quad \sum_{i\in\mathcal{A}^{\mathrm{d}}} w_i \left(\chi_i^c \frac{1}{c_i} + \chi_i^0 - D'_i\right)^+ \tag{P2a}$$

subject to:

$$c_i = \Gamma_i \nu_i, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{P2b}$$

$$\sum_{i\in\mathcal{A}^{\mathrm{d}}} \Gamma_i \nu_i \leq N, \tag{P2c}$$

$$c_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{P2d}$$

$$\nu_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{P2e}$$

Problem (P2) is a straightforward weighted tardiness minimization under a capacity constraint. The objective function (P2a) is the sum of all the tardiness terms associated to the various applications, each multiplied by the corresponding weight. The set of constraints (P2b) specifies the number of cores available per VM, while constraint (P2c) enforces that no more than the remaining N cores are allocated to soft deadline applications. In the end, constraints (P2d)-(P2e) state that all the variables c_i and ν_i are taken from the natural numbers, as expected given their interpretation.

Taking the continuous relaxation of Problem (P2), it is possible to consider Karush-Kuhn-Tucker (KKT) conditions both necessary and sufficient for optimality, given the fact that Slater's constraint qualification holds and the objective function is convex. The following Theorem 2 provides a closed form optimal solution for the continuous problem.

Theorem 2. *Let $\mathcal{A}^t \subseteq \mathcal{A}^d$ be the set of tardy applications and, in order to simplify the notation, call $\bar{\mathcal{A}}^t = \mathcal{A}^d \setminus \mathcal{A}^t$. For the sake of simplicity, assume application 1 belongs to $\mathcal{A}^t$. Furthermore, let $\bar{c}_i : \chi_i^c \frac{1}{\bar{c}_i} + \chi_i^0 = D'_i$. The optimal solution of the continuous relaxation of Problem* (P2) *is:*

$$\begin{cases} c_j = \bar{c}_j, & \forall j \in \bar{\mathcal{A}}^t, \\ c_i = \sqrt{\dfrac{w_i}{w_1}\dfrac{\chi_i^c}{\chi_1^c}}\, c_1, & \forall i \in \mathcal{A}^t \setminus \{1\}, \\ c_1 = \dfrac{N - \sum_{j\in\bar{\mathcal{A}}^t} \bar{c}_j}{1 + \sum_{i\in\mathcal{A}^t\setminus\{1\}} \sqrt{\frac{w_i}{w_1}\frac{\chi_i^c}{\chi_1^c}}}, \end{cases} \tag{8}$$

with:

$$\nu_i = \frac{c_i}{\Gamma_i}, \quad \forall i \in \mathcal{A}^d. \tag{9}$$

Proof. See Appendix A. □

Theorem 2 only provides a solution for the continuous relaxation of Problem (P2). Additionally the ML model used to formulate tardiness and, consequently, the objective function (P2a) is just a first approximation for the system performance. These reasons motivate a search procedure to determine an integer solution and, possibly, leverage the higher accuracy of simulation-based models to attain more efficiency or a decrease in global weighted tardiness. Algorithm 2 performs this solutions space exploration: it receives in input the set of soft deadline applications and the maximum number of iterations allowed to run, then it returns the optimized VMs allocation. The detailed description of its behavior follows.

Algorithm 2 Resource rebalancing

Require: $\mathcal{A}^{\mathrm{d}}, \bar{k} \in \mathbb{N}$
1: $\mathcal{A}^{\mathrm{t}} \leftarrow \mathcal{A}^{\mathrm{d}}$
2: **repeat**
3: compute $\mathbf{c}$ and $\boldsymbol{\nu}$ as per (8) and (9)
4: remove all $i \in \mathcal{A}^{\mathrm{t}} : c_i \geq \bar{c}_i$
5: **until** $\mathcal{A}^{\mathrm{t}}$ does not change
6: round $\boldsymbol{\nu}$ and make the configuration feasible
7: **repeat** $k \leftarrow 1, 2, \ldots$
8: $\boldsymbol{\nu}^{\mathrm{old}} \leftarrow \boldsymbol{\nu}$
9: $L \leftarrow \emptyset$
10: **for** $i \in \mathcal{A}^{\mathrm{d}}$ **do**
11: **for** $j \in \mathcal{A}^{\mathrm{d}} : j \neq i$ **do**
12: choose $\Delta\nu_i$ and $\Delta\nu_j$ so that $\Gamma_i\Delta\nu_i = \Gamma_j\Delta\nu_j$
13: evaluate δ_{ij}^{H} with hyperbola, as per (4)
14: **if** $\delta_{ij}^{\mathrm{H}} < 0 \wedge \nu_j - \Delta\nu_j \geq 1$ **then**
15: add $(\Delta\nu_i, \Delta\nu_j)$ to L
16: **end if**
17: **end for**
18: **end for**
19: $\delta_{ij}^* \leftarrow 0$
20: $m^* \leftarrow (0, 0)$
21: **for** $(\Delta\nu_i, \Delta\nu_j) \in L$ **do**
22: evaluate δ_{ij}^{L} with Lundstrom
23: **if** $\delta_{ij}^{\mathrm{L}} < \delta_{ij}^*$ **then**
24: $m^* \leftarrow (\Delta\nu_i, \Delta\nu_j)$
25: **end if**
26: **end for**
27: apply m^* to $\boldsymbol{\nu}$
28: **until** $k = \bar{k} \vee \boldsymbol{\nu} = \boldsymbol{\nu}^{\mathrm{old}}$
29: **return** $\boldsymbol{\nu}$

First of all, the loop at lines 2–5 obtains the initial continuous solution via the formulas. Since $\mathcal{A}^{\mathrm{t}}$ is not known in advance, Algorithm 2 repeatedly applies the formulas (8)-(9) and removes from $\mathcal{A}^{\mathrm{t}}$ the applications that are not tardy (see Appendix A for additional discussion). Right after computing the solution for the continuous relaxation, the VMs allocation $\boldsymbol{\nu}$ is heuristically made feasible at line 6. The adopted approach is simple: Each ν_i is rounded to the next greater integer and, if needed, one VM from the applications with the smallest weights is removed until the capacity constraint (P2c) is satisfied.

At line 7 starts the main loop of the local search algorithm. Each iteration amounts to the exploration of a neighborhood centered around the current optimal solution, with a best improvement policy. The neighborhood contains all the configurations that can be reached if an application gives up some VMs and its cores are reassigned to another application, see line 12. Different applications are possibly hosted on different VMs, then it is necessary to make sure that these moves respect the proportionality of virtual central processing units. For example, if job $\bar{\imath}$ yields 1 quad-core VM and $\bar{\jmath}$ is hosted on single-core instances, the latter will receive 4 replicas, thus leaving unchanged the total resource pool. Since the size of the neighborhood is $O(|\mathcal{A}^{\mathrm{d}}|^2)$, relying solely on simulations for the exploration would make optimization times too long for use at run time. This is the reason why the neighborhood is first swept via the algebraic approximation (4) in the two

nested loops at lines 10–18. Instead of reevaluating the full objective function for each move, the algorithm only considers the partial effect on the weighted tardiness values for i and j: these terms are called δ_{ij}. According to the condition at line 14, only the promising moves that do not take all the VMs from an application ($\nu_j - \Delta\nu_j \geq 1$) and improve the overall weighted tardiness in the hyperbolic approximation ($\delta_{ij}^{\mathrm{H}} < 0$) are later assessed via simulations.

The second part of the main iteration (lines 19–27) goes over all the promising moves identified with the algebraic approximation and reevaluates them via the simulation with Lundstrom. The variable m^*, initially set to the empty move, keeps track of the best improving move. At last, the main cycle concludes by applying the best move and increasing the iteration count. Line 28 states the stopping criterion: either the algorithm keeps exploring for the maximum number of iterations $\bar{k}$, or the latest neighborhood search did not provide any improving moves, meaning that the search procedure reached a local minimum.

5 Experimental Results

The aim of this section is the analysis of the accuracy and of the scalability of the algorithms when used in the deployment of real systems.

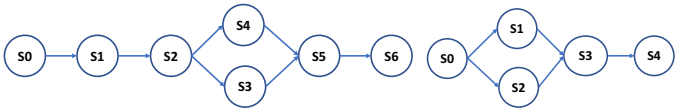
This section is organized as follows: in Section 5.1, the experimental setup is introduced, then the results of OPT_IC and OPT_JR are described in Section 5.2 and Section 5.3, respectively. Finally, Section 5.4 presents an example of the usage of the proposed algorithms in a real scenario.

5.1 Experimental Setup

The validation of the algorithms is based on TPC-DS[2], a standard industry benchmark for evaluating the performance of data warehouse and big data systems, which is based on a SQL-like workload. Multiple experiments were conducted and four cluster configurations were considered to verify that the proposed algorithms behave consistently on all the scenarios. The four considered cluster configurations are the following:

- *A3*: the benchmarks were deployed on the Microsoft Azure cluster with the HDInsight Platform as a Service (PaaS) offering based on Spark 1.6.2 release and Ubuntu 14.04[3] and exploiting A3 VMs. Each VM includes 4 cores with 7 GB of RAM and 250 GB local disk. The configuration of the workers consisted of up to 48 cores. Each Spark executor had 2 cores with 2 GB RAM.
- *D12v2*: a Microsoft Azure cluster with the HDInsight PaaS offering based on Spark 1.6.2 release and Ubuntu 14.04, but based on the D12v2 VMs. Each VM includes 4 cores, 28 GB of RAM, and 200 GB local SSD. The number of cores of workers was varied between 12 and 52. Each Spark executor had 2 cores with 8 GB RAM.
- *D13v2*: a Microsoft Azure cluster with the HDInsight PaaS offering based on Spark 1.6.2 release and Ubuntu 14.04, but based on the D13v2 VMs. Each VM includes 8 cores, 56 GB of RAM, and 400 GB local SSD. The number of cores of workers was 48. Each Spark executor had 8 cores with 40 GB RAM.
- *P8*: an in-house cluster based on IBM POWER 8 (P8) available at Politecnico di Milano. It is based on Spark 1.4.1 running on Red Hat 7.3 and includes six VMs with 11 cores and 60 GB of RAM each. Fiber channel disks up to 12 TB of physical storage were available. Spark executors were configured with 2 cores and 4 GB RAM while 8 GB were allocated to the driver. Workers runs were supported by four VMs and used between 6 and 44 cores.

2. http://www.tpc.org/tpcds/
3. https://azure.microsoft.com/en-us/services/hdinsight/



(a) Q26 (b) Q52

Figure 3. Directed acyclic graphs

Table 3
Data set sizes considered during collection of data about real execution time of queries on clusters.

Query	A3 [GB]	D12v2 [GB]	D13v2 [GB]	P8 [GB]
Q20	-	-	1000	-
Q26	500	500	-	500 - 1000
Q40	500	500	-	500 - 1000
Q52	500	500	1000	500 - 1000
Q55	500	500	1000	500 - 1000

For what concerns the master, on the Azure deployments two dedicated master nodes were used and the Spark driver had 4 GB allocated, while on P8 the driver was running on a dedicated master node with the same VM configuration (11 cores and 60 GB of RAM).

To validate the proposed algorithms the data about the execution time of different queries from TPC-DS with input datasets of 500 GB, and 1000 GB on the different clusters were collected. For each size, a dataset was generated through the TPC-DS generator. Table 3 summarizes the combinations of queries, dataset sizes, and clusters which were considered during the collection of real data. For every combination, the profiling process was repeated ten times, considering that the profiles collect statistical information about applications.

As an example, the DAGs of Q26 and Q52 on Azure at 500 GB are presented in Figure 3. The execution of OPT_IC has been supported by a server equipped with two Intel Xeon E5530 2.40 GHz processors with 48 GB of RAM, while OPT_JR has been executed on a server with two Intel Xeon Silver 4114 2.20 GHz processors with 48 GB of RAM.

5.2 OPT_IC Validation

This subsection presents the results of the validation of OPT_IC by considering the following goals: i) the behavior of the algorithm is investigated (in terms of number of iterations, execution time and objective function trend) when the deadline is reduced, ii) the error in estimating the required number of cores with different initial deadline is evaluated, and, finally, iii) the accuracy of the interpolation based on the database values is assessed.

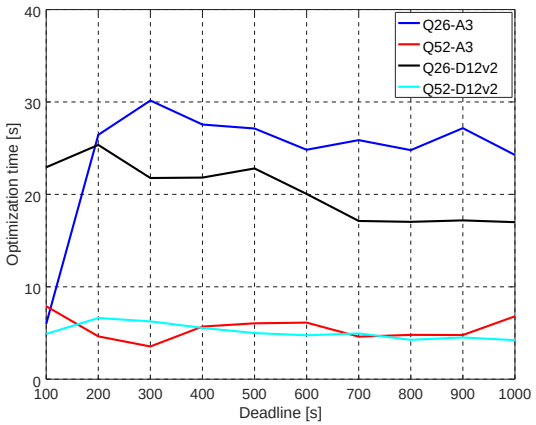


Figure 4. OPT_IC execution times for Q26 (500 GB) and Q52 (500 GB) for Azure deployments

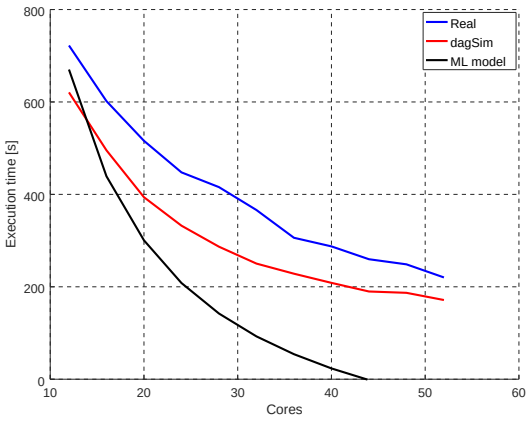


Figure 6. Mean values of real execution time, of ML model approximation, and dagSim simulation for Q26 (500 GB) executed on D12v2

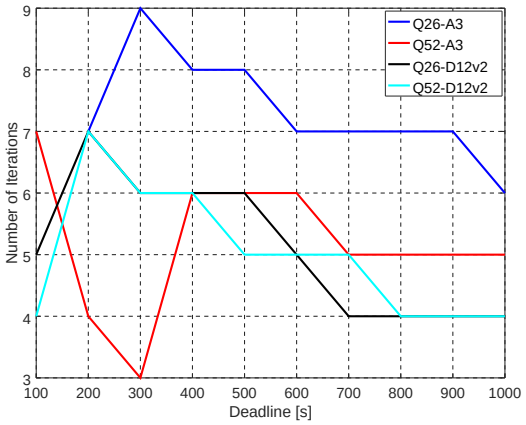


Figure 5. OPT_IC number of iterations for Q26 (500 GB) and Q52 (500 GB) on Azure deployments

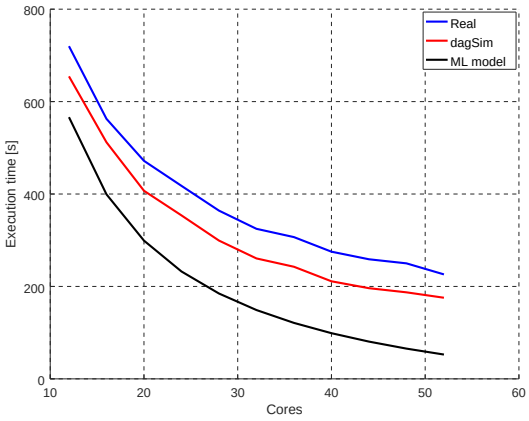


Figure 7. Mean values of real execution time, of ML model approximation, and dagSim simulation for Q52 (500GB) executed on D12v2

Tests Description

The analysis is performed by providing smaller and smaller deadlines to verify that more and more resources are allocated. Moreover, the smaller is the deadline, the harder is the optimization problem to be solved (since a feasible solution might not exist), so tightening deadlines can lead to a larger execution time for OPT_IC. The deadline values varied in the range between 100 and 1000s with 100s step. The analysis includes the execution time, the number of iterations, and the value of the objective function.

Analysis of the Behavior of the Algorithm

Figure 4 reports the optimization times required for Q26 and Q52 for both Azure deployments (the dataset size is 500 GB). It is interesting to observe that for Q26 the optimization time does not significantly increase even when the problem becomes infeasible because of the deadline reduction. Figure 5 reports the number of iterations needed to find the optimal solution. The range of iterations for all the four combinations is between 3 and 9, and Q26 on A3 is the query for which the algorithm requires the largest number of iterations (9). The number of iterations is small even if, as shown in Figure 7, there is a significant difference between the estimations produced by the ML model, which are used during initialization phase of OPT_IC, and the estimations of dagSim, which are used during the actual optimization process. So, even if the initial solution is far from the correct solution, since the machine learning models are not accurate, up to 82% mean absolute percentage error (MAPE) for Q40 on P8 machines, the algorithm is able in a few iterations to converge to the final optimal solution.

Figure 8 shows that the execution time for the optimization process for P8 infrastructure is significantly larger compared to Azure, since the number of tasks that should be evaluated is larger and *dagSim*'s execution time grows, justifying the usage of a cache to store already computed simulation results. Moreover, the error of the ML model for this cluster (82% MAPE) is significantly larger than the maximum error of the ML models for the Azure deployments (14% MAPE for Q26 on A3 VMs, 53% for Q52 on D12v2). Nevertheless, as shown in Figure 9, the number of iterations of OPT_IC does not present a large increase. This confirms how, even if the starting point identified through the ML model prediction is very far from the exact solution, the algorithm is still able to fast move towards accurate solutions.

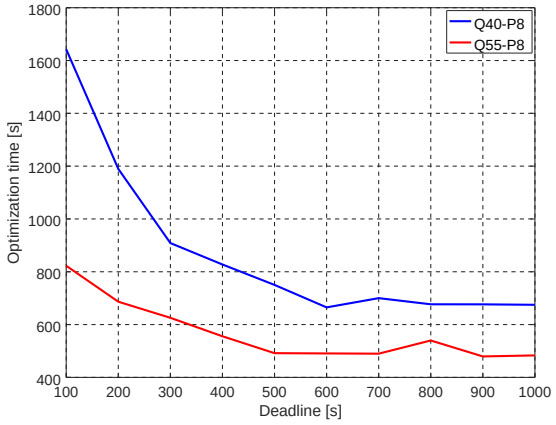


Figure 8. OPT_IC execution times for Q40 (1000 GB) and Q55 (1000 GB) for P8 deployments

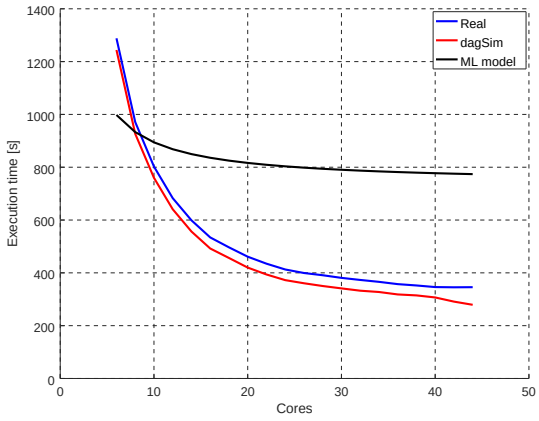


Figure 11. Mean values of real execution time, of ML model approximation and dagSim simulation for Q55 (1000 GB) executed on P8

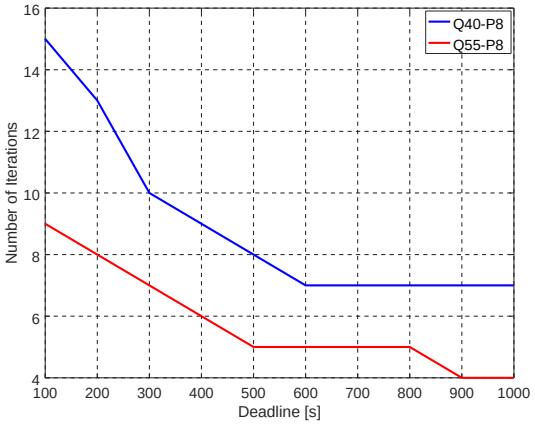


Figure 9. OPT_IC number of iterations for Q40 (1000 GB) and Q55 (1000 GB) on P8

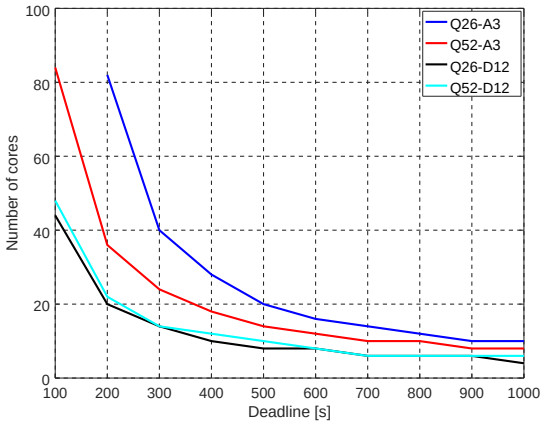


Figure 12. Deployment of Q26 and Q52 on A3 and D12v2 VM types

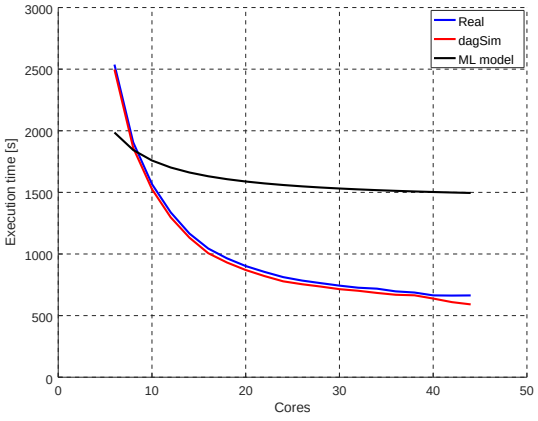


Figure 10. Mean values of real execution time, of ML model approximation and dagSim simulation for Q40 (1000 GB) executed on P8

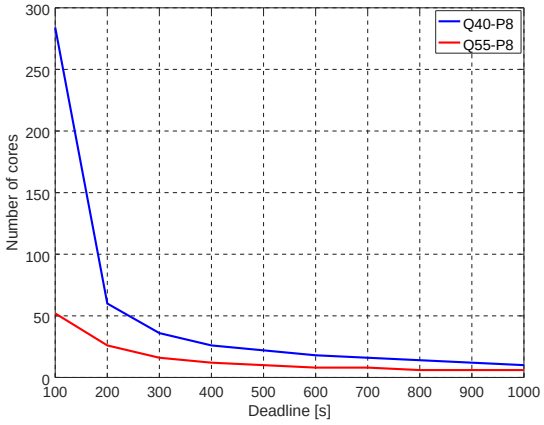


Figure 13. Deployment of Q40 (500 GB) and Q55 (500 GB) on P8 infrastructure

Table 4
OPT_IC prediction validation of Q26 (500 GB) and Q52 (500 GB) on D12v2

Query	D [ms]	c^{r}	c^{p}	ϵ [%]
Q26	280,550	24	28	−14.28
Q26	186,067	36	44	−18.18
Q26	158,288	48	52	−8.33
Q52	276,791	24	32	−25.00
Q52	175,395	36	48	−25.00
Q52	150,489	48	56	−14.28

Table 5
OPT_IC prediction validation with Q40 (1000 GB) and Q55 (1000 GB) on P8

Query	D [ms]	c^{r}	c^{p}	ϵ [%]
Q40	2,537,015	6	6	0.00
Q40	1,905,821	8	8	0.00
Q40	1,566,992	10	10	0.00
Q40	1,337,868	12	12	0.00
Q40	1,165,666	14	14	0.00
Q40	1,044,557	16	16	0.00
Q40	964,752	18	18	0.00
Q40	902,173	20	20	0.00
Q40	854,692	22	22	0.00
Q40	812,298	24	24	0.00
Q40	784,445	26	24	7.69
Q40	763,447	28	26	7.14
Q40	743,283	30	28	6.67
Q40	726,324	32	30	6.25
Q40	718,641	34	30	11.76
Q40	696,641	36	34	5.56
Q40	687,299	38	34	10.53
Q40	664,080	40	40	0.00
Q40	662,526	42	40	4.76
Q40	663,823	44	40	9.09
Q55	1,288,271	6	6	0.00
Q55	971,212	8	8	0.00
Q55	802,620	10	10	0.00
Q55	682,972	12	12	0.00
Q55	598,917	14	14	0.00
Q55	534,180	16	16	0.00
Q55	496,608	18	16	11.11
Q55	461,197	20	18	10.00
Q55	434,755	22	20	9.09
Q55	412,768	24	22	8.33
Q55	399,407	26	22	15.38
Q55	390,949	28	24	14.29
Q55	380,981	30	24	20.00
Q55	373,125	32	24	25.00
Q55	365,936	34	26	23.53
Q55	357,285	36	28	22.22
Q55	352,222	38	28	26.32
Q55	346,128	40	30	25.00
Q55	345,276	42	30	28.57
Q55	345,712	44	30	31.82

Finally, Figure 12 and Figure 13 show how the obtained results present the expected characteristics. The tightening of the deadline produces an increment of the number of cores. Moreover, the relationship between the deadline and the required number of cores is not linear, but depicts a convex curve.

Validation

To evaluate the quality of OPT_IC solutions, the prediction error ϵ in estimating the required number of cores to satisfy the deadline as the main metric is analyzed. ϵ is defined as follows:

$$\epsilon = \frac{c^{\mathrm{r}} - c^{\mathrm{p}}}{c^{\mathrm{r}}} \tag{10}$$

where c^{r} is the real number of cores to run within an a priori deadline D identified by inspecting the application logs and c^{p} is the number estimated by OPT_IC.

For the D12v2 deployment, six different cases obtained by varying deadline and query were considered for the validation on the Azure cloud. Table 4 shows the obtained results: the maximum obtained error is 25%.

Table 5 shows the results of the validation on the P8 infrastructure by comparing the real number of cores versus the predicted value. The deadlines have been set equal to the actual execution time for each possible value of number of cores. The error ranges between 0.00% and 11.76% for the first query (Q40) and between 0.00% and 31.82% for the second one (Q55). It is worth noting that, even if all the presented predictions are conservative (i.e., the number of cores is overestimated), this property cannot be considered as general. Indeed, in the considered range of number of cores, OPT_IC makes conservative predictions since dagSim is also conservative. On the contrary, a non-conservative performance estimation of dagSim (i.e., underestimating execution time like in 40 cores case) can cause non-conservative predictions by OPT_IC.

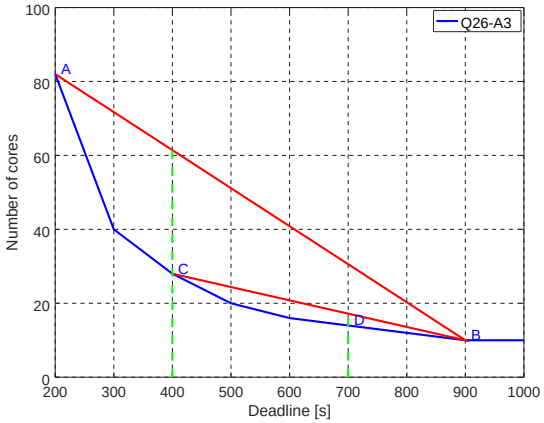


Figure 14. Prediction of the optimal number of cores for fulfilling a deadline constraint using OPT_IC and interpolation approach

Off-line Interpolation Accuracy

As discussed previously, Figures 12 and 13 show the results of OPT_IC for different combinations of queries and architectures. The y-axis represents the number of cores needed to fulfill the deadline reported in the x-axis (varied with 100s step). It can be noticed how for different queries and for different deployments, the curves depicting the number of cores vs. the deadline constraints are always convex. For this reason, the linear interpolation implemented by the optimization service when a specific deadline constraint is not available in the system history lookup table (see Section 2) is conservative with respect to the predictions of OPT_IC. In the following, an illustrative example will be discussed.

Let us consider Figure 14 where the number of cores required by Q26 when it is executed on A3 VMs (blue curve) is shown and let us initially assume that OPT_IC has been executed for only two points, A (deadline set at 200 s) and B (deadline set at 900 s), i.e., the information about all the other deadlines is not yet available in the lookup table. The red line connecting A and B shows the applied linear interpolation on the only two available data. Let us assume that the query needs to be run with a 400 s deadline: since this deadline is not available in the lookup table (i.e., the required number of core for this deadline has not yet been estimated), the linear interpolation returns 62 cores as result with an error of 138%. On the contrary, the optimization service computes off line that only 28 cores are needed (point C). Then, let us assume that in the following the query is run with 700-seconds deadline. As for the previous deadline, the linear interpolation is calculated, but this time instead of point A and B (the closest) points C and B are selected. In this case, as shown in Figure 14, the predicted number of cores is 18, while OPT_IC determines off-line 14 cores minimum cost configuration with a lower percentage error equal to 29%.

Since the optimization process is time consuming, this example shows how the strategy to compute the minimum cost configuration for a few points during application profiling is effective. Indeed, the accuracy of the linear interpolation improves when additional runs of the same query are performed on the production system.

5.3 OPT_JR validation

This section describes the settings and the configurations used to validate the re-balancer tool and presents the obtained results. As for OPT_IC, the results are analyzed in terms of the execution time of the algorithm, and of the obtained value of the objective function. Moreover, the size of the list of candidates L of Algorithm 2 is also considered, since it is related to the number of executions of the performance prediction service Lundstrom, which is time consuming. Furthermore, the results achieved by the parallel implementation of the algorithm aimed at reducing OPT_JR execution time are discussed.

Tests Description

In the following, the experimental setup for the validation of the OPT_JR tool on a set of six test cases is detailed. All the tests use the queries presented in Table 3 and target the P8 cluster. The maximum number of iterations is set to 10. Each test is characterized by the pressure p defined as:

$$p = \frac{c_{\min}}{N} \tag{11}$$

where c_{min} is the minimum number of cores which does not cause a deadline violation (obtained through OPT_IC) and N is the number of available cores.

Table 6 describes Test1, which is the base test and includes four queries. Table 7 details the difference of Test2, Test3, Test4 with respect to Test1. In Test2 N is increased to get pressure equal to 0.66: there are more available cores than required. On the contrary, in Test3 N is decremented to set the pressure equal to 2.00: the number of available cores is half of the required.

Table 6
OPT_JR model validation Test1. All weights set to 1

Query	M	m	V	v	D [s]
Q26	28	8	4	2	1,000
Q52	28	8	4	2	1,000
Q40	56	18	4	2	1,000
Q55	56	18	4	2	1,000

Table 7
Summary of changes of OPT_JR validation tests with respect to Test1

Test	N (500 GB)	N (1000 GB)	p	Weight of Q52
Test1	24	44	1.33	1
Test2	48	88	0.66	1
Test3	16	28	2.00	1
Test4	24	44	1.33	10

In Test4 the weight of Q52 is set to 10 to show how the algorithm actually considers weights in the computation of the solution. Each test has been run with two different configurations: in the first the dataset of all the queries has been set to 500 GB, in the second the dataset of all the queries has been set of 1000 GB. On the contrary, Test5 is a more complex test whose details are shown in Table 8: multiple instances of the same query are indeed considered so that the overall number of running application is 10. The dataset size of all the queries have been set to 500 GB for the first instance and to 1000 GB for the second one. Finally, Test6 uses the same applications of Test5 with both the configurations at the same time (i.e., all the queries of Table 8 are added twice). N has been set to 78 in the first instance of Test5, 150 in the second instance of Test5, and 226 in the only instance of Test6.

Analysis of the Behavior of the Algorithm

This section presents the experimental results obtained with OPT_JR on the tests presented above. By comparing the parameters and the results for Test1, Test2, and Test3, it is possible to see that by reducing the number of available cores, both the number of iterations (see, e.g., Figure 15) and the execution time (see Table 10) tend to increase. This was expected, since the smaller number of available cores (in

Table 8
OPT_JR model validation Test5

Query	M	m	V	v	D [s]
Q40	56	18	4	2	600
Q55	56	18	4	2	800
Q26	28	8	4	2	800
Q40	56	18	4	2	800
Q52	28	8	4	2	800
Q55	56	18	4	2	800
Q26	28	8	4	2	1,000
Q40	56	18	4	2	1,000
Q52	28	8	4	2	1,000
Q55	56	18	4	2	1,000

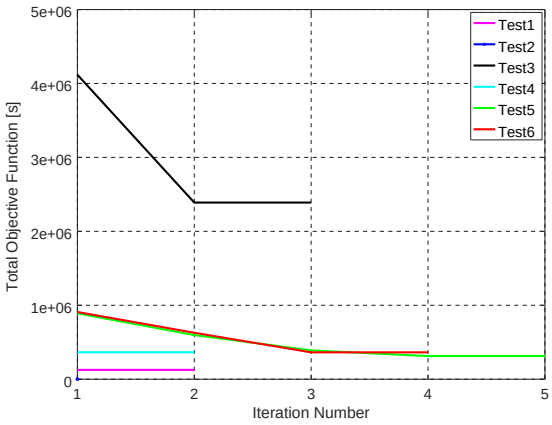


Figure 15. OPT_JR validation (1000 GB): Total Objective Function vs. Iterations number

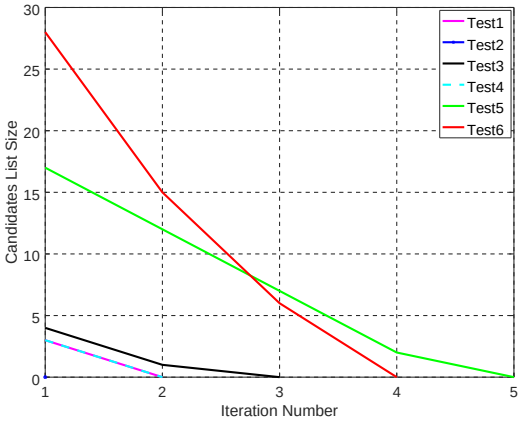


Figure 16. OPT_JR validation: size of Candidates List L vs. Iteration number

Test3) makes the problem more difficult to be solved. For Test2 a single point is reported: in the first iteration, the objective function is always 0 since the tardiness of all the queries is 0.

The curves reporting the values of the total objective functions (see Figure 15) lower down progressively on each iteration proving that the overall local search is able to improve the initial heuristic assignment by selecting the best core configuration switch at each iteration. In all the tests the algorithm stops before reaching the maximum number of iterations (10) since the Candidate List L becomes empty, demonstrating that OPT_JR identifies in few iterations the final local optimum solution. Since no change is actually applied during the last iteration, there is no improvement in the objective function in the final step.

Finally, the results about the candidates list L size are presented in Figure 16. This value plays a crucial role to understand the performance of the algorithm since the size of the list corresponds to the number of times the predictor is invoked (a time-consuming operation since it implies, among different things, a call to the Lundstrom tool as an external process). Moreover, the size of L list has a monotonic decreasing behavior with respect to the number of algorithm iteration. Also in this graph, only one point is reported for Test2: since the initial solution already satisfies all the deadlines, no further change is evaluated.

Table 9
Number of cores assigned to Q52

Test	c
Test1	8
Test2	12
Test3	4
Test4	12

Table 10
OPT_JR execution times and Speedup (SU) with different number of threads

		ST	MT(2)		MT(4)	
Test	Size	Time [s]	Time [s]	SU	Time [s]	SU
Test1	500	13.73	12.18	1.13	8.86	1.55
Test1	1000	34.86	25.61	1.36	20.31	1.72
Test2	500	13.88	11.64	1.19	8.96	1.55
Test2	1000	19.21	14.34	1.34	11.34	1.69
Test3	500	13.78	11.65	1.18	8.83	1.56
Test3	1000	45.70	34.91	1.31	29.43	1.55
Test4	500	13.77	11.62	1.19	9.33	1.48
Test4	1000	34.09	25.66	1.33	20.33	1.68
Test5	500	60.86	48.99	1.24	42.23	1.44
Test5	1000	224.89	160.75	1.40	128.40	1.75
Test6	-	313.04	229.82	1.36	181.10	1.73

Table 9 shows the different number of cores assigned to Q52 in different tests when the dataset size is set to 1000 GB and how the different parameters can impact on the identified solution. The minimum number of cores necessary to satisfy the deadline ($1000s$) would be 12. In Test1, since the number of available cores is less than the required (p is 1.33), not all the required resources can be assigned. For this reason, OPT_JR assigns to Q52 only 8 cores. On the contrary, Test2 is characterized by $p < 1$ (i.e., there are more resources than required), so all the necessary cores (12) can be given to Q52. Test3 is characterized by $p = 2.00$: for this reason, the number of cores of Q52 is even lower than in Test1 (4 vs. 8). Finally, in Test4 there are not enough resources to satisfy all the requirements. Nevertheless, since Q52 has a larger weight than the other queries receives all the required cores (12).

Performance Evaluation

OPT_JR exploits a set of *OpenMP* directives [4] to execute the re-balancer in multi-thread mode. The objective of using this approach consists of exploring with a parallel process the local neighborhood in such a way that the Lundstrom performance predictor can be invoked in a multi-thread way over multiple candidates.

Table 10 shows the speed-up performance obtained on Test1 - Test5 (both with dataset size equal to 500 GB and to 1000 GB) and Test6. Specifically, the tests were executed with *Lundstrom* predictor in single and multi-thread mode (using 2 and 4 cores). The usage of more than 4 processors does not provide significant advantages on these tests because

4. https:/http://www.openmp.org/

Table 11
Description of the case study scenario

Query	Dataset Size [GB]	W	Submission Time	D [s]
Q20	1000	5	t_0	780
Q52	1000	1	$t_1 = t_0 + 320s$	600
Q55	1000	1	$t_1 = t_0 + 320s$	600

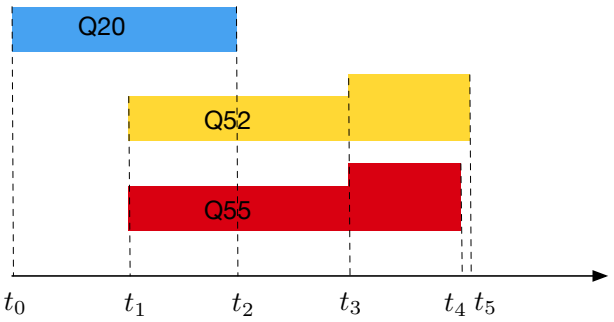


Figure 17. Usage of resources of queries run in the case study

of the limited size of the candidate move list L (see Figure 16). The above results suggest that the proposed approach is adequate to manage at run-time real clusters supporting long running batch queries as the time requested to obtain a new configurations is in the range of a few minutes.

5.4 Real Scenario Case Study

This section presents the behavior of OPT_IC and of OPT_JR in a real use case. The considered scenario consists of three queries (Q20, Q52, and Q55) submitted for execution at different time instants on a 6 D13v2 VMs cluster with 1000 GB dataset and a total number of cores $N = 48$. All the queries have been initially executed separately to collect the execution logs to be exploited by dagSim.

The scenario details are presented in Table 11. Initially (t_0), only Q20 is submitted for execution on the cluster with its deadline set to 780 seconds. OPT_IC estimates 16 cores (corresponding to two VMs) to fulfill the deadline. Since the full cluster is initially available, all the required VMs can be assigned to Q20. After about 320 seconds (t_1), two new queries (Q52 and Q55) are submitted with deadline set to 600s for both of them. OPT_IC estimates that 32 cores (4 VMs) should be assigned to each query to satisfy the deadline. Since the cluster has not enough capacity, the system is in heavy load condition. OPT_JR is triggered to minimize the weighted tardiness by rebalancing VMs across queries: in the identified solution two VMs are assigned to each query. It is worth noting that with this configuration, Q20 (which has the largest weight) is expected to meet its deadline while Q52 and Q55 do not.

After about 250 seconds (t_2), Q20 ends its execution. Nevertheless, the system is still in heavy load condition: the execution of Q52 and Q55 was slowed down by the limited number of cores, so each query would require more than the initially estimated number of cores. For this reason OPT_JR is executed again to rebalance cores between Q52 and Q55. The tardiness of the two queries is similar, so OPT_JR assigns to each them half of the available resources (24 cores, 3 VMs). The detection of the end of Q20 and the successive invocation of OPT_JR takes only some seconds, but the implementation of the suggested solution (i.e., booting and update of the Spark configuration) takes some minutes. At time $t_3 = t_2 + 340$ the new resources are actually available to Q52 and Q55 and the queries run with three VMs each from now on. When Q55 ends (t_4), Q52 is still running and its end is foreseen in few seconds. For this reason the number of its executors is not incremented, since the overhead of the rescaling can nullify the benefits of the incremented resources. Finally, in few seconds also Q52 ends so that the presented scenario is completed.

To evaluate the accuracy of OPT_JR, the value of its objective function (i.e., the overall weighted tardiness) is compared with the actual value at each its invocation. At t_1, the contributions to the objective functions are provided only by Q52 and Q55 since Q20 is expected to end before its deadline. The estimated objective function is $908s$ while its actual value is $924s$. The obtained error is very low (1.73%) thanks to the good accuracy of the simulator. It is worth noting that both in the computation of the estimated and of the real objective function, Q52 and Q55 are assumed to be always executed with the resources allocated to them at the moment. Since, more resources will then be made available to them, the final value of the sum of the tardiness will be smaller.

The second invocation of OPT_JR occurs at t_2: the estimated tardiness value is $320s$ while the actual value is $406s$ so that the obtained error is 21%. The larger error is mainly due by the delay in the assignment of the new resources. OPT_JR is assuming that they will be available at t_2, but they were available only at t_3. This delay causes also the postponing of the end of the execution of Q52 and of Q55, resulting in having tardiness larger than estimated for both the queries. Nevertheless, the use of OPT_IC and of OPT_JR allows the user to satisfy the deadline of the query Q20 with the higher weight and to minimize the tardiness of the others.

6 Related Work

Provisioning and scheduling resources for big data applications in cloud infrastructures face several challenges such as dynamicity of queries, load fluctuation, performance unpredictability, and heterogeneity of resources. One of the main challenges for big data cluster frameworks is how to partition and dynamically allocate the resources to reach high efficiency and scalability. Resource partitioning and dynamic allocation mechanisms, indeed, are enablers for providing efficient resource provisioning and improve system utilization. Recently, significant work has been performed to address these issues decoupling also the resource management from the programming model. A number of technologies have been proposed, such as YARN [13], Mesos [14], Omega [15], and Borg [16].

AROMA [17] is one of the first frameworks for the automated resource allocation and configuration of MapReduce clusters. AROMA mines historical execution data in order to profile past submissions and match incoming jobs to the available past signatures for predicting the performance. In this way, the proposed system can avoid deadline violations stated in Service Level Agreements (SLAs) and minimize the costs, with an average percentage error on completion estimations around 12%.

More recently, Delimitrou et al. proposed Paragon [18] an online and scalable scheduler for large-scale datacenters, which uses collaborative filtering techniques in order to characterize an unknown incoming application. Paragon handles resource assignment and leverages information from previous application runs and offline training in order to cope with performance and interference effects on the application execution.

The Paragon engine has been extended in Quasar [19] to estimate the impact of horizontal (more servers) and vertical (more resources per server) scaling. The system handles both resource allocation and assignment. If the performance deviates from the SLAs constraints, Quasar reclassifies the workload and adjusts the allocation and/or the assignment decisions to meet the application deadlines minimizing the used resources. If on one hand Paragon and Quasar are very general and take explicitly heterogeneity and workloads interference into account, the frameworks require an exhaustive offline profiling of some reference applications and to monitor and to profile the incoming application for several minutes on separate servers before running on production. Vice versa, the solutions proposed in this work require only to consider few (two or three) runs to obtain comparable performance and are able to minimize soft applications tardiness in case of heavy load.

Similar ideas have been implemented by Spark inventors in Hemingway [20], which, however, is specialized in the identification of the optimal cluster configuration for Spark MLlib based applications. Hemingway takes into account some machine learning algorithm peculiarities (e.g., how the convergence rate may be affected by the cluster size), adopts experiment design to collect as few training points as possible, and achieves an average prediction error under 20%.

In [21] the authors propose CCRP, a framework aimed at increasing the data center utilization by assigning complementary jobs to different resource types. Resources to jobs assignment is formulated as a classical integer linear programming problem with the goal to maximize cluster utilization. The problem is then heuristically solved by assigning complementary jobs (whose demands on multiple resource types are complementary to each other) to the same VM to increase the resource utilization. Jobs are classified in short and long running through a non-linear classifier. Moreover, an opportunistic allocation scheme is adopted and unused resources of short jobs (estimated through a deep neural network) are reallocated to other jobs. Authors demonstrated that their approach can achieve 50% higher resource utilization with respect to state of the art solutions, however no deadline guarantees can be provided to jobs execution.

Finally, Alipourfard et al. [22] 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, driving the search to improve the prediction accuracy of those configuration, which are close to the best for a specific deadline. This work is suitable to optimize recurrent workloads i.e., identifies only the minimum capacity to provide to an application to complete within the same deadline. Vice versa, our approach can manage competing applications under scarce resources and has demonstrated to provide good performance prediction and deadline achievement capabilities across multiple settings.

7 Conclusions & Future Work

In this paper, optimization policies for the run-time management of Spark applications on cloud clusters have been proposed. Two main problems have been considered: (i) how to determine the minimum capacity to devote to an application to fulfill a deadline, and (ii) under heavy load, how to rebalance system capacity to minimise the weighted tardiness for soft deadline applications. Simulation-optimization heuristics have been developed. Starting from an initial solution obtained through a relaxed non-linear programming model, the solutions space is efficiently explored by exploiting machine learning, approximated analytical techniques, and simulation to estimate application performance. In this way, a favorable trade-off between application performance prediction accuracy and solution algorithms running times can be obtained.

A comprehensive experimental validation proved how the approach is effective to manage resources of both private and public cloud clusters. Results demonstrated that the average percentage error of the proposed resources allocation with respect to the real optimal solution is around 8%. Moreover, complex problems like computing the optimal redistribution of resources among tens of applications can be tackled in few minutes.

Future work will extend the optimization framework to support the resource provisioning of continuous applications integrating batch and streaming workloads. Moreover, clusters including GPU resources to support advanced deep learning Spark applications will be also considered.

Acknowledgments

The results of this work have been partially funded by EUBra-BIGSEA (grant agreement no. 690116), a Research and Innovation Action funded by the European Commission under the Cooperation Programme, Horizon 2020 and the Ministério de Ciência, Tecnologia e Inovação, RNP/Brazil (grant GA0000000650/04).

Eugenio Gianniti is also partially supported by 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, 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] D. Henschen. (2016) Spark Summit East Report: Enterprise Appeal Grows. [Online]. Available: https://www.constellationr.com/blog-news/spark-summit-east-report-enterprise-appeal-grows

[3] (2017, Mar.) Worldwide semiannual Big Data and analytics spending guide. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=IDC_P33195

[4] Forbes, "Data Mining adoption in Target Industries," https://blogs.forbes.com/louiscolumbus/data-mining-adoption/, 2016, online; accessed 3 May 2018.

[5] Datameer, "3 Top Big Data Use Cases in Financial Services. How Financial Services Companies are Gaining Momentum in Big Data Analytics and Getting Results," https://www.datameer.com/wp-content/uploads/2018/02/big-data-use-case-financial-services.pdf, 2016, online; accessed 3 May 2018.
[6] ——, "Five New Big Data Use Cases for 2018 – Healthcare Precision Medicine," https://www.datameer.com/blog/five-new-big-data-use-cases-2018-part-3/, 2018, online; accessed 3 May 2018.
[7] The digital universe in 2020. [Online]. Available: http://idcdocserv.com/1414
[8] J. Kross and H. Krcmar, "Model-based performance evaluation of batch and stream applications for Big Data," in *MASCOTS Proc.*, 2017.
[9] D. Ardagna, E. Barbierato, A. Evangelinou, E. Gianniti, M. Gribaudo, T. B. M. Pinto, A. Guimarães, A. P. C. da Silva, and J. M. Almeida, "Performance Prediction of Cloud-Based Big Data Applications," in *ICPE*, 2018.
[10] 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*, 2016.
[11] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic resource inference and allocation for MapReduce environments," in *Proceedings of the Eighth International Conference on Autonomic Computing*, Jun. 2011.
[12] 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. [Online]. Available: http://doi.acm.org/10.1145/2788402.2788410
[13] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *SOCC 2013 Proc.*, 2013.
[14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488
[15] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *SIGOPS European Conference on Computer Systems (EuroSys)*, Prague, Czech Republic, 2013, pp. 351–364. [Online]. Available: http://eurosys2013.tudos.org/wp-content/uploads/2013/paper/Schwarzkopf.pdf
[16] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *EuroSys 2015 Proc.*, 2015.
[17] P. Lama and X. Zhou, "AROMA: automated resource allocation and configuration of mapreduce environment in the cloud," in *ICAC 2012 Proc.*, 2012, pp. 63–72.
[18] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," 2013.
[19] ——, "Quasar: Resource-efficient and qos-aware cluster management," in *ASPLOS 2014 Proc.*, 2014.
[20] X. Pan, S. Venkataraman, Z. Tai, and J. Gonzalez, "Hemingway: Modeling distributed optimization algorithms," *CoRR*, vol. abs/1702.05865, 2017.
[21] J. Liu, H. Shen, and H. S. Narman, "Ccrp: Customized cooperative resource provisioning for high resource utilization in clouds," in *IEEE Big Data 2016 Proc.*, 2016.
[22] 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.

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.

Enrico Barbierato earned a MSc and a PhD in Computer Science from University of Turin (Italy) and a second MSc in Advanced Studies in Artificial Intelligence from the Katholieke Universiteit of Leuven (Belgium). His professional experience in the private sector since 1992 concerned Finance, Telecom and Energy&Utilities markets. Enrico joined Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico of Milan in 2016 as a temporary researcher and teaching assistant. His initial research activity regarded the study of abductive expert systems and deadlock avoidance strategies in scenarios deploying autonomous guided vehicles, to later shift to the performance evaluation of multiformalism models and applications of Markovian Agents.

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 data-intensive applications hosted on Clouds, as well as in the performance modeling of such systems via both simulation and analytical methods.

Marco Lattuada received the Master and the Ph.D. degrees in Computer Engineering from Politecnico di Milano, Italy, in 2006 and 2010 respectively. In 2012 and in 2013 he was visiting researcher at European Space Agency. Since 2010, he has been temporary researcher and lecturer at Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano. His research interests include methodologies for performance estimation of big data applications running on cloud cluster, methodologies for High Level Synthesis, and methodologies for performance estimation and automatic generation of code for multiprocessor embedded heterogeneous architectures.

Appendix A
Proof

Proof of Theorem 2.

Proof. The objective function (P2a) is irregular, but introducing $\bar{\mathbf{c}}$ and exploiting constraints (P2b) it is possible to give the following regular formulation:

$$\min_{\mathbf{c}} \sum_{i \in \mathcal{A}^{\mathrm{d}}} w_i \left(\chi_i^c \frac{1}{c_i} + \chi_i^0 - D_i' \right) \tag{P3a}$$

subject to:

$$\sum_{i \in \mathcal{A}^{\mathrm{d}}} c_i \leq N, \tag{P3b}$$

$$c_i \leq \bar{c}_i, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{P3c}$$

$$c_i \geq 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{P3d}$$

Constraints (P3c) enforce that no application receives more resources than needed to achieve zero tardiness, hence contributions to the objective function remain nonnegative as expected.

The Lagrangian of Problem (P3) is given by:

$$\begin{aligned} \mathcal{L}(\mathbf{c}) = & \sum_{i \in \mathcal{A}^{\mathrm{d}}} w_i \left(\chi_i^c \frac{1}{c_i} + \chi_i^0 - D_i' \right) + \\ & + \lambda^N \left(\sum_{i \in \mathcal{A}^{\mathrm{d}}} c_i - N \right) + \\ & + \sum_{i \in \mathcal{A}^{\mathrm{d}}} \lambda_i^{\bar{c}} (c_i - \bar{c}_i) + \\ & - \sum_{i \in \mathcal{A}^{\mathrm{d}}} \lambda_i^c c_i \end{aligned} \tag{12}$$

and stationarity conditions lead to:

$$\frac{\partial \mathcal{L}}{\partial c_i} = -\frac{w_i \chi_i^c}{c_i^2} + \lambda^N + \lambda_i^{\bar{c}} - \lambda_i^c = 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{13}$$

while complementary slackness conditions are:

$$\lambda^N \left(\sum_{i \in \mathcal{A}^{\mathrm{d}}} c_i - N \right) = 0, \quad \lambda^N \geq 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{14a}$$

$$\lambda_i^{\bar{c}} (c_i - \bar{c}_i) = 0, \quad \lambda_i^{\bar{c}} \geq 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{14b}$$

$$\lambda_i^c c_i = 0, \quad \lambda_i^c \geq 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{14c}$$

$c_i > 0$ by the definition of tardiness: thanks to the conditions (14c), it holds $\lambda_i^c = 0, \ \forall i \in \mathcal{A}^{\mathrm{d}}$. From the KKT (13) descends:

$$\lambda^N + \lambda_i^{\bar{c}} = \frac{w_i \chi_i^c}{c_i^2}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{15}$$

In heavy load conditions, $\sum_{i \in \mathcal{A}^{\mathrm{d}}} \bar{c}_i > N$. Such inequality implies that $\exists \hat{\imath} : c_{\hat{\imath}} < \bar{c}_{\hat{\imath}}$, then by (14b) it holds $\lambda_i^{\bar{c}} = 0$, which, added to (15), yields $\lambda^N > 0$. In particular, this implies that constraint (P3b) is active in every optimal solution.

The optimal solution features a set of applications with strictly positive tardiness, equivalently defined as $\mathcal{A}^{\mathrm{t}} = \left\{ i \in \mathcal{A}^{\mathrm{d}} : c_i < \bar{c}_i \right\}$. As already observed for $\hat{\imath}$, $\lambda_i^{\bar{c}} = 0, \ \forall i \in \mathcal{A}^{\mathrm{t}}$. Exploiting (15), it holds that:

$$\frac{w_i \chi_i^c}{c_i^2} = \frac{w_j \chi_j^c}{c_j^2}, \quad \forall (i,j) \in \mathcal{A}^{\mathrm{t}} \times \mathcal{A}^{\mathrm{t}} \tag{16}$$

and:

$$c_i = c_1 \sqrt{\frac{w_i}{w_1} \frac{\chi_i^c}{\chi_1^c}}, \quad \forall i \in \mathcal{A}^{\mathrm{t}}. \tag{17}$$

Let $\bar{\mathcal{A}}^{\mathrm{t}} = \mathcal{A}^{\mathrm{d}} \setminus \mathcal{A}^{\mathrm{t}}$, hence $c_i = \bar{c}_i, \ \forall i \in \bar{\mathcal{A}}^{\mathrm{t}}$. Substituting into constraint (P3b) taken as equality:

$$N = \sum_{i \in \bar{\mathcal{A}}^{\mathrm{t}}} \bar{c}_i + \sum_{i \in \mathcal{A}^{\mathrm{t}} \setminus \{1\}} \sqrt{\frac{w_i}{w_1} \frac{\chi_i^c}{\chi_1^c}} c_1 + c_1, \tag{18}$$

whence comes, via simple algebraic transformations, the formula for c_1 that completes the optimal system (8). □