A Game-Theoretic Approach for Runtime Capacity Allocation in MapReduce

Eugenio Gianniti, Danilo Ardagna, Michele Ciavotta
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Milan, Italy
Email: name.surname@polimi.it

Mauro Passacantando
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
Email: mauro.passacantando@unipi.it

***Abstract*—Nowadays many companies have available large amounts of raw, unstructured data. Among Big Data enabling technologies, a central place is held by the MapReduce framework and, in particular, by its open source implementation, *Apache Hadoop*. For cost effectiveness considerations, a common approach entails sharing server clusters among multiple users. The underlying infrastructure should provide every user with a fair share of computational resources, ensuring that service level agreements (SLAs) are met and avoiding wastes.**

In this paper we consider mathematical models for the optimal allocation of computational resources in a Hadoop 2.x cluster with the aim to develop new capacity allocation techniques that guarantee better performance in shared data centers. Our goal is to get a substantial reduction of power consumption while respecting the deadlines stated in the SLAs and avoiding penalties associated with job rejections. The core of this approach is a distributed algorithm for runtime capacity allocation, based on Game Theory models and techniques, that mimics the MapReduce dynamics by means of interacting players, namely the central Resource Manager and Class Managers.

***Index Terms*—Hadoop; Capacity Allocation; Admission Control; Generalized Nash Equilibrium Problem.**

I. Introduction

A large number of enterprises currently commits to the extraction of information from huge data sets as part of their core business activities. Applications range from fraud detection to one-to-one marketing, encompassing business analytics and support to decision making in both private and public sectors. In order to cope with the unprecedented amount of data and the need to process them in a timely fashion, new technologies are increasingly adopted in the industry, following the *Big Data* paradigm. Among such technologies, Apache Hadoop is already widespread and predictions suggest a further increase in its future adoption. IDC estimates that, by 2020, nearly 40% of Big Data analyses will be supported by public Clouds [1].

Apache Hadoop is an open source software suite that enables the elaboration of vast amounts of data on clusters of commodity hardware. Hadoop implements the MapReduce paradigm, automatically ensuring parallelization, distribution, fault-tolerance, reliability, and monitoring. In order to obtain a high level of scalability, Hadoop 2.x overcomes previous versions drawbacks by implementing a distributed resource management system, with a central Resource Manager (RM) that provides resources for computation to Application Masters (AMs) entitled to manage the submitted jobs.

Despite the convenience of this paradigm and the undeniably widespread adoption of Hadoop within the IT industry, still there are no tools that support developers and operators in achieving optimal capacity planning of MapReduce applications. In this context the main issue [2], [3] is that the execution time of a MapReduce job is generally unknown in advance: for some systems, capacity allocation can become a critical aspect. Moreover, resource allocation policies need to decide job execution and rejection rates in a way that users' workloads can meet their service level agreements (SLAs) and the overall cost is minimized.

This paper investigates the theoretical foundations for the optimal runtime management of cluster resources in private Clouds. Precisely, we focus on the joint admission control and capacity allocation problem, seeking to fulfill SLAs while minimizing energy-related costs. Overall, ICT energy demand sums up to 7% of the world consumption and it was expected to rise up to 12% by 2017 [4], with a further tendency towards a shift from devices to networks and data centers consumption [5]. Indeed, worldwide ICT systems account for 2–4% of global CO_2 emissions and it is expected that they can reach up to 10% in 5–10 years [6].

We propose a theoretical approach in which the allocation problem is solved periodically based on a prediction of the forthcoming system load. In particular, we adopt Game Theory techniques, which found successful application in the field of Cloud computing [7]–[10], and use them to provide a distributed, scalable solution to the joint admission control and capacity allocation of multi-class Hadoop clusters.

This paper is organized as follows. In Section II, we state the problem at hand and relevant assumptions. Section III presents a preliminary, centralized mathematical programming formulation, whilst Section IV builds on it to propose a distributed game-theoretic model. Then we analyze our results in Section V, whilst Section VI discusses related work. In the end, Section VII wraps up and draws conclusions.

II. Problem Statement and Design Assumptions

Figure 1 shows the reference technological system, featuring the Hadoop 2.x framework running on a private Cloud. The

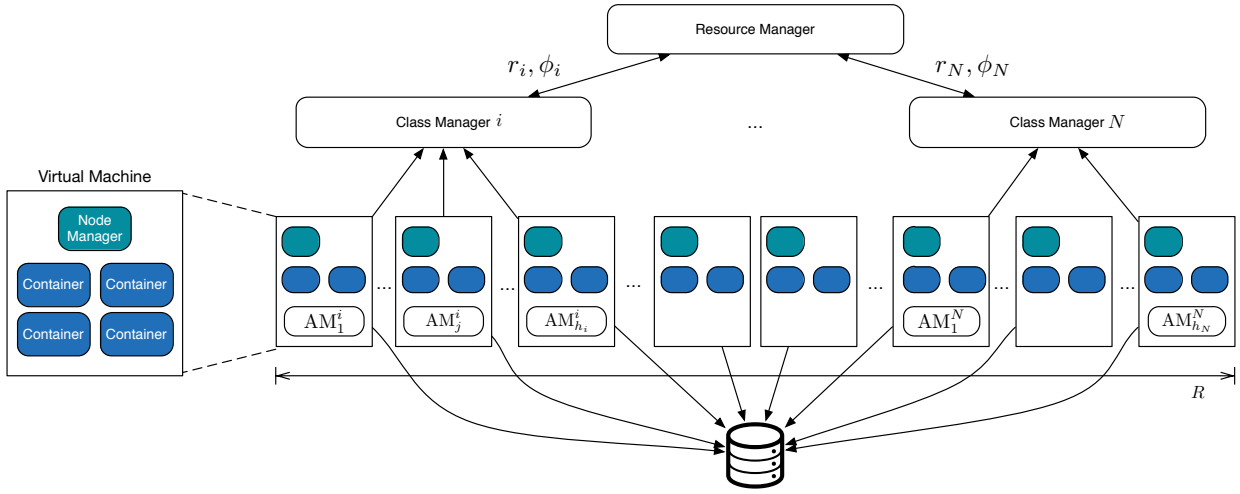

Figure 1: Reference technology

private Cloud cluster supports several user classes competing for resources, which are managed via the YARN Capacity Scheduler. Each class collects similar jobs, i.e., applications that share analogous values for parameters characterizing their performance: they have the same *job profile*. Following the advancements published in [3], [11], job profiles include the number of tasks involved in both the Map and Reduce phase, as well as their average and maximum duration.

The modeled cluster supports the concurrent execution of a maximum of R virtual machines (VMs), which we assume homogeneous for the sake of simplicity. In order to allow for elasticity, the reference system does not store data on the Hadoop Distributed File System (HDFS), as this could lead to data corruption or poor performance: on the contrary, according to the practice suggested by major Cloud providers, data resides on external storage. With this approach, storage and computation are completely decoupled, so that it is possible to vary resource allocation over time without the risk of data loss or time consuming file system rebalancing.

According to our vision of a novel resource allocation policy, every application class is managed by a Class Manager (CM), which negotiates the required resources with a central RM, entitled to split the available capacity among submitted jobs. The set of application classes is denoted with $\mathcal{A}$ and $N = |\mathcal{A}|$. For all CMs $i \in \mathcal{A}$, the RM assigns r_i VMs. In other words, in this scenario the proposed framework acts as the YARN Capacity Scheduler [12], assigning every application class i to a separate queue and providing a portion ϕ_i of the total resources, where:

$$\phi_i \triangleq \frac{r_i}{\sum_{j=1}^{N} r_j}, \ \forall i \in \mathcal{A}.$$

Given $\bar{\rho}$, the time unit cost to run a single VM, it is possible to obtain the total cost of execution as $\sum_{i=1}^{N} \bar{\rho} r_i$.

For every application class i, an SLA establishes that a maximum of H_i^{up} jobs can be executed concurrently. However, the system can autonomously decide to reject a portion of such jobs upon payment of a penalty. Finally, the accepted h_i jobs cannot be fewer than H_i^{low} and the system commits to complete them within a deadline D_i. We denote with $\mathcal{P}_i(h_i)$ the penalty functions associated to the possible rejection of some jobs. They are assumed to be decreasing and convex: this is reasonable as it means that penalties increase at least linearly in the number of rejected jobs.

According to the obtained number of resources r_i, a CM may need to reject some jobs, then it proceeds to activate a suitable number of AMs to coordinate the admitted ones. In this scenario, the AMs have the only duty of managing the resources obtained by the CMs so as to carry out the associated job tasks, without directly taking part in the allocation process.

We propose to solve our problems hourly, based on a prediction of the load H_i^{up}, to dynamically reallocate resources among application classes, while also avoiding the overhead and costs of booting and shutting down VMs too frequently.

In the Hadoop framework each computational node hosts several slots that execute Map and Reduce tasks. In particular, according to the YARN configuration, computational resources are split in containers, so that every VM can be used to concurrently run c_i^M Map or c_i^R Reduce tasks, according to their requirements. These parameters depend only on the job classes owing to the assumption of homogeneity made on VMs. The total number of containers assigned to an application class for Map and Reduce tasks is represented by s_i^M and s_i^R, respectively. Again, these variables give a simple representation of the workload required to complete jobs in each class due to the homogeneity assumption on VMs. Precisely, with s_i^M and s_i^R we represent the number of Map and Reduce tasks that run concurrently, hence the maximum size of each wave.

As previously stated, according to [3], it is possible to derive from the Hadoop logs a job profile, i.e., a set of parameters that characterize the execution of jobs in each class. In this paper we use a more refined formulation, as in [11]. The estimated minimum and maximum execution times are computed with

Table I: Centralized Model Parameters

$\mathcal{A}$	Set of job classes
N	Number of CMs, or $\lvert\mathcal{A}\rvert$
$\bar{\rho}$	Time unit cost for running a VM in the cluster
H_i^{up}	Maximum concurrency required in the SLA for job class i
H_i^{low}	Minimum concurrency required in the SLA for job class i
ψ_i^{low}	Reciprocal of H_i^{up}
ψ_i^{up}	Reciprocal of H_i^{low}
R	Total capacity of the cluster in number of VMs
A_i	Coefficient associated to Map tasks in job profile i, [11]
B_i	Coefficient associated to Reduce tasks in job profile i, [11]
E_i	Constant term in job profile i, [11]
c_i^M	Map containers supported on one VM for job class i
c_i^R	Reduce containers supported on one VM for job class i
α_i	Slope of the penalty term $\mathcal{P}_i(\psi_i)$ for job class i
β_i	Constant term of the penalty term $\mathcal{P}_i(\psi_i)$ for job class i

Table II: Centralized Model Decision Variables

r_i	Number of VMs assigned for the execution of job class i
h_i	Number of jobs concurrently executed in job class i
ψ_i	Reciprocal of the concurrency degree h_i
s_i^M	Number of Map slots assigned for the execution of job class i
s_i^R	Number of Reduce slots assigned for the execution of job class i

formulae similar to the following:

$$T_i = A_i \frac{h_i}{s_i^M} + B_i \frac{h_i}{s_i^R} + C_i. \tag{1}$$

The coefficients A_i, B_i, and C_i properly aggregate the job profile parameters, which are measured directly by Hadoop and easily obtainable from its logs. These formulae are used to predict job execution times, given the amount of allocated resources and the concurrency level. As shown in [11], the relative error of such predictions is between 10 and 19%. Equation (1) can be used to derive deadline constraints, which can be formulated as:

$$T_i = A_i \frac{h_i}{s_i^M} + B_i \frac{h_i}{s_i^R} + C_i \leq D_i, \quad \forall i \in \mathcal{A}. \tag{2}$$

where D_i are the deadlines. In the following, we adopt the parameter $E_i = C_i - D_i$. Notice that, by definition, it holds $E_i < 0$, as nonnegative values would mean that jobs of class i cannot be completed on time.

In light of the above, we can say that the ultimate goal of the proposed approach is to determine the optimal values of h_i, s_i^M, s_i^R, and r_i so that the sum of costs and rejection penalties is minimized, while the deadlines set by SLAs are met. In Table I are reported all the parameters used in the models discussed in the subsequent sections, while in Table II we summarize the decision variables.

III. Mathematical Programming Formulation

Building upon the observations and assumptions previously discussed, we formulate a preliminary mathematical programming model that formalizes the problem. The model is the following:

$$\min_{\mathbf{r},\mathbf{h},\mathbf{s}^M,\mathbf{s}^R} \quad \sum_{i=1}^{N} \bar{\rho} r_i + \sum_{i=1}^{N} \mathcal{P}_i(h_i) \tag{P1a}$$

subject to:

$$\sum_{i=1}^{N} r_i \leq R, \tag{P1b}$$

$$H_i^{low} \leq h_i \leq H_i^{up}, \quad \forall i \in \mathcal{A}, \tag{P1c}$$

$$\frac{A_i h_i}{s_i^M} + \frac{B_i h_i}{s_i^R} + E_i \leq 0, \quad \forall i \in \mathcal{A}, \tag{P1d}$$

$$\frac{s_i^M}{c_i^M} + \frac{s_i^R}{c_i^R} \leq r_i, \quad \forall i \in \mathcal{A}, \tag{P1e}$$

$$r_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1f}$$

$$h_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1g}$$

$$s_i^M \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1h}$$

$$s_i^R \in \mathbb{N}, \quad \forall i \in \mathcal{A}. \tag{P1i}$$

In problem (P1) the objective function (P1a) has a term representing the cost of executing all the assigned VMs and another for penalties. Constraint (P1b) ensures that the cluster capacity bounds the total assigned resources. Further, the set of constraints (P1c) imposes the minimum and maximum job concurrency levels, according to the SLAs. Similarly, constraints (P1d) exploit the job profiles to ensure the deadlines are met. Constraints (P1e) guarantee that every application class receives enough VMs to support the number of containers they should run concurrently. The left hand side is a conservative estimate of the resources needed to support at the same time s_i^M and s_i^R containers: this expression greatly simplifies the analysis. Constraints (P1f)–(P1i) require all the variables to be nonnegative integers, as expected for their interpretation.

Since the optimization problem is nonlinear due to the constraints family (P1d) and the penalty terms $\mathcal{P}_i(h_i)$, it is advisable to study its continuous relaxation. Nonetheless, the solutions to the proposed models have to be integer, as it is only possible to instantiate a discrete number of VMs, then we will discuss a heuristic approach to the issue in Section IV-D.

Moreover, constraints (P1d) are not convex, thus ruling out many important results for nonlinear optimization. We address this issue introducing a new set of variables $\psi_i \triangleq h_i^{-1}$, so that constraints (P1d) become convex: with this change of variables it is possible to write a convex nonlinear problem.

Now, let us assume that the penalties are linear in the new variables ψ_i, hence it is possible to write them as $\alpha_i \psi_i - \beta_i$, $\forall i \in \mathcal{A}$. The corresponding penalty term in the objective function (P1a) is then $\mathcal{P}_i(h_i) = \alpha_i h_i^{-1} - \beta_i$, $\forall i \in \mathcal{A}$. This expression is consistent with the assumptions of convexity and monotonicity made on $\mathcal{P}_i(h_i)$. The formulation reads:

$$\min_{\mathbf{r},\boldsymbol{\psi},\mathbf{s}^M,\mathbf{s}^R} \quad \sum_{i=1}^{N} \bar{\rho} r_i + \sum_{i=1}^{N} (\alpha_i \psi_i - \beta_i) \tag{P2a}$$

subject to:

$$\sum_{i=1}^{N} r_i \leq R, \tag{P2b}$$

$$\psi_i^{low} \leq \psi_i \leq \psi_i^{up}, \quad \forall i \in \mathcal{A}, \tag{P2c}$$

$$\frac{A_i}{s_i^M \psi_i} + \frac{B_i}{s_i^R \psi_i} + E_i \leq 0, \quad \forall i \in \mathcal{A}, \tag{P2d}$$

$$\frac{s_i^M}{c_i^M} + \frac{s_i^R}{c_i^R} \leq r_i, \quad \forall i \in \mathcal{A}, \tag{P2e}$$

$$r_i \geq 0, \quad \forall i \in \mathcal{A}, \tag{P2f}$$

$$\psi_i \geq 0, \quad \forall i \in \mathcal{A}, \tag{P2g}$$

$$s_i^M \geq 0, \quad \forall i \in \mathcal{A}, \tag{P2h}$$

$$s_i^R \geq 0, \quad \forall i \in \mathcal{A}. \tag{P2i}$$

Following the proposed change of variables, constraints (P1c) become constraints (P2c) where $\psi_i^{low} = (H_i^{up})^{-1}$ and $\psi_i^{up} = \left(H_i^{low}\right)^{-1}$. Further, as can be seen from constraints (P2f)–(P2i), we take the continuous relaxation of the otherwise mixed integer problem. Thanks to the typically high values of s_i^M, s_i^R, and r_i, it is possible to round the real solution without affecting too much the optimal value.

Problem (P2) is convex and Slater constraint qualification holds: the Karush-Kuhn-Tucker (KKT) conditions are, then, necessary and sufficient for optimality. Relying on the KKT conditions, it is possible to prove[1] Proposition III.1 and simplify the problem formulation:

Proposition III.1. *The optimal values attained by s_i^M, s_i^R, and ψ_i in problem* (P2) *are:*

$$s_i^M = \xi_i^M r_i, \quad \forall i \in \mathcal{A}, \tag{3a}$$

$$s_i^R = \xi_i^R r_i, \quad \forall i \in \mathcal{A}, \tag{3b}$$

$$\psi_i = K_i r_i^{-1}, \quad \forall i \in \mathcal{A}, \tag{3c}$$

where:

$$\xi_i^M \triangleq \frac{c_i^M}{1 + \sqrt{\frac{B_i}{A_i}\frac{c_i^M}{c_i^R}}}, \quad \forall i \in \mathcal{A}, \tag{4a}$$

$$\xi_i^R \triangleq \frac{c_i^R}{1 + \sqrt{\frac{A_i}{B_i}\frac{c_i^R}{c_i^M}}}, \quad \forall i \in \mathcal{A}, \tag{4b}$$

$$K_i \triangleq -\frac{\left(\sqrt{\frac{A_i}{c_i^M}} + \sqrt{\frac{B_i}{c_i^R}}\right)^2}{E_i}, \quad \forall i \in \mathcal{A}. \tag{4c}$$

Building upon the results of Proposition III.1, it is possible to define two new sets of parameters, r_i^{up} and r_i^{low}, which are the optimal number of resources needed to complete jobs of class i on time at maximum or minimum concurrency level, respectively:

$$r_i^{up} = \frac{K_i}{\psi_i^{low}} = K_i H_i^{up}, \quad \forall i \in \mathcal{A}, \tag{5a}$$

$$r_i^{low} = \frac{K_i}{\psi_i^{up}} = K_i H_i^{low}, \quad \forall i \in \mathcal{A}. \tag{5b}$$

However, problem (P2) forces to centralize on a single node data characterizing different application classes. This approach would have been natural in the first release of Hadoop where a single Job Tracker was in charge of both assigning resources to jobs and ensuring they were carried out. Instead, Hadoop 2.x distributes these duties among the RM and the AMs, so a more appealing approach would be to set them apart and solve distinct problems. In order to address this issue, we propose a distributed formulation where each CM and the RM solve smaller local problems. The decision variables are split among all these entities and the bargaining for resources becomes a game involving $N + 1$ players.

[1]For the complete proof, refer to [13].

Table III: Distributed Model Parameters

ρ_i^{up}	Maximum bid CM i can place to obtain resources
$\hat{\rho}$	Maximum price the RM can set, $\max_i \rho_i^{up}$
r_i^{up}	Optimal VMs to sustain job class i at maximum concurrency
r_i^{low}	Optimal VMs to sustain job class i at minimum concurrency
p_i	Virtual penalty paid for each VM short of r_i^{up}
ξ_i^M	Optimal number of Map containers to execute per VM
ξ_i^R	Optimal number of Reduce containers to execute per VM
K_i	Optimal number of VMs to carry out one job of class i

Table IV: Distributed Model Decision Variables

ρ_i^a	Bid placed by CM i to obtain resources
ρ	Time unit price set by the RM for one VM
y_i	Logical variable stating whether CM i offers a high enough bid

IV. Game Theoretic Formulation

Given the reference scenario presented in Section II and Figure 1, we propose a game among the RM and CMs. Iteratively, the RM assigns VMs to the CMs, while they determine the concurrency level and the optimal distribution of slots for Map and Reduce tasks on the acquired resources. If the current assignment is not satisfactory, each CM can offer to pay a higher price for extra resources, then the RM assigns VMs to the highest bidders.

This mechanism naturally leads to making r_i decision variables of the RM problem, whilst s_i^M, s_i^R, and ψ_i are decision variables of the CM problems. Conversely, s_i^M, s_i^R, and ψ_i are parameters for the RM problem, like r_i for each CM problem.

Remarkably, the formulation that is to be laid out in this section belongs to the category of generalized Nash equilibrium problems (GNEPs) [14]. These are a generalization of classical Nash games where each player's strategy set is not fixed, but depends on the strategies adopted by the opponents. Indeed, the RM affects CMs' strategy sets by imposing an allowance of VMs and, in turn, CMs influence the RM with bids for resources.

In this section we introduce some more parameters and decision variables, summarized in Tables III and IV. Nonetheless, the parameters and variables already presented in Tables I and II are still of interest in the following models.

A. Class Managers

Class Managers are responsible for negotiating with the RM the assignment of a suitable number of VMs to carry out the submitted jobs. This behavior is modeled with a problem where r_i, the number of assigned VMs, are parameters, i.e., no CM can change their value. Therefore, the model features only constraints (P2c), (P2d), and (P2e), while its objective

function takes into account only the penalty term of (P2a). Conversely, the RM problem might end up not being aware of the penalties applied to job rejections, then we introduce a new set of variables, ρ_i^a. These have the interpretation of bids placed by the CMs to virtually "buy" resources from the RM, but this aspect will be clarified in Sections IV-B and IV-C. For the time being, it is sufficient to know that ρ_i^a cannot be lower than the cost of execution $\bar{\rho}$, nor greater than some given upper bounds ρ_i^{up}. The CM problem can be formulated as follows:

$$\min_{\psi_i, \rho_i^a, s_i^M, s_i^R} \quad \alpha_i \psi_i - \beta_i \tag{P3a}$$

subject to:

$$\bar{\rho} \leq \rho_i^a \leq \rho_i^{up}, \tag{P3b}$$

$$\psi_i^{low} \leq \psi_i \leq \psi_i^{up}, \tag{P3c}$$

$$\frac{A_i}{s_i^M \psi_i} + \frac{B_i}{s_i^R \psi_i} + E_i \leq 0, \tag{P3d}$$

$$\frac{s_i^M}{c_i^M} + \frac{s_i^R}{c_i^R} \leq r_i, \tag{P3e}$$

$$s_i^M \geq 0, \tag{P3f}$$

$$s_i^R \geq 0, \tag{P3g}$$

$$\psi_i \geq 0, \tag{P3h}$$

$$\rho_i^a \geq 0. \tag{P3i}$$

Recall that, in this class of problems, r_i play the role of parameters, since they are not controlled by CMs. Thus, constraint (P3e) sets a bound on the possible values attained by s_i^M and s_i^R, which in turn affect ψ_i. Due to this, the set of strategies each CM can play depends on the strategy adopted by the RM and is indirectly influenced by other CMs strategies.

Applying the techniques discussed in Section III, it is possible to obtain formally the same results of Proposition III.1 for Problem (P3) as well. Such formulae can be used to solve in closed form the admission control problem.

B. Resource Manager

The Resource Manager receives from all the CMs requests for a certain number of VMs. Its role is to allocate the available resources, trying to satisfy all the requests to guarantee the timely completion of the jobs. Consequently, r_i are the only decision variables. On the other hand, the model requires new variables: ρ and y_i. Notice that, in this case, we have just one instance with variables subscripted over the whole set $\mathcal{A}$, contrasting to the formulation proposed in Section IV-A, where the model appears in N instances separately solved by CMs.

The virtual pricing mechanism introduced to distribute the original problem is centered on the variables ρ and y_i and is inspired by the pricing policy enforced by Amazon for the allocation of *spot* VMs. These are a class of VMs that can be acquired without previous agreements, but do not provide any availability guarantees to customers. Indeed, it is possible to apply for a number of such VMs offering to pay a unit price. Based on the received offers and the current availability of resources, Amazon sets a single spot price. Now, customers who offered enough get their share of spot VMs paying only the spot price, which can be lower than their bids. However, Amazon may at any moment terminate them raising the spot price to a value that is higher than the received bids.

In our setting, the spot price is ρ and is set by the RM. Every time the RM solves its problem, it determines the price based on the bids ρ_i^a placed by the CMs. Every CM that offers more than ρ is allowed to receive VMs up to the optimal upper bound r_i^{up}. On the other hand, CMs placing a bid lower than ρ will receive exactly their minimum resource share, i.e., r_i^{low}. The parameters obtained in equations (5) define for r_i a range where job starvation and waste of resources are avoided. The variables y_i play their role in determining, through proper constraints, whether CM i placed a bid greater than ρ or not. In any case, the VMs unit price is ρ, for both overbidding and underbidding CMs.

Clearly, to implement this behavior, it is necessary to use as objective function the total "revenue" and maximize it. However, in this way the model has no information about penalties. As long as the cluster capacity is not saturated, this is not much of an issue, since the tendency is to assign as many VMs as possible up to a value that cannot exceed r_i^{up}. A more appealing objective function should have a term related to the job rejection penalties in addition to the one expressing the virtual revenue. A similar penalty term should discourage the assignment of less resources than needed, but at the same time should not foster an uncontrolled growth of r_i when there is no need to add more VMs. With these issues in mind, we introduce a new set of parameters, p_i, which are an equivalent penalty coefficient normalized on the lacking resources.

The complete model for the RM is the following:

$$\max_{\mathbf{r}, \mathbf{y}, \rho} \quad \sum_{i=1}^{N} (\rho - \bar{\rho})\, r_i - \sum_{i=1}^{N} p_i \left(r_i^{up} - r_i\right) \tag{P4a}$$

subject to:

$$\sum_{i=1}^{N} r_i \leq R, \tag{P4b}$$

$$r_i \geq r_i^{low}, \quad \forall i \in \mathcal{A}, \tag{P4c}$$

$$r_i \leq \left(r_i^{up} - r_i^{low}\right) y_i + r_i^{low}, \quad \forall i \in \mathcal{A}, \tag{P4d}$$

$$\bar{\rho} \leq \rho \leq \hat{\rho}, \tag{P4e}$$

$$\rho - \rho_i^a \leq M\,(1 - y_i)\,, \quad \forall i \in \mathcal{A}, \tag{P4f}$$

$$\rho_i^a - \rho \leq M y_i, \quad \forall i \in \mathcal{A}, \tag{P4g}$$

$$r_i \geq 0, \quad \forall i \in \mathcal{A}, \tag{P4h}$$

$$y_i \in \{0, 1\}, \quad \forall i \in \mathcal{A}, \tag{P4i}$$

$$\rho \geq 0. \tag{P4j}$$

In the objective function (P4a), the first term quantifies the virtual gain obtained by the RM for the resources it assigns to CMs. Instead, the second term expresses the penalty contribution. Constraints (P4c) and (P4d) are the lower and upper bounds on the resources assigned to each CM. In particular, the upper bound is r_i^{up} when CM i offers more than the price set by the RM, otherwise it is r_i^{low}, thus forcing $r_i = r_i^{low}$. This behavior is obtained through constraints (P4f) and (P4g), which are examples of the so called "big M" constraints; they make sure that y_i has value 1 when CM i offers enough, 0 otherwise. Constraints (P4e) ensure that the

Algorithm IV.1 Best Reply

1: $r_i \leftarrow r_i^{low},\ \forall i \in \mathcal{A}$
2: $s_i^M \leftarrow s_i^{M,low},\ \forall i \in \mathcal{A}$
3: $s_i^R \leftarrow s_i^{R,low},\ \forall i \in \mathcal{A}$
4: $\psi_i \leftarrow \psi_i^{up},\ \forall i \in \mathcal{A}$
5: $\rho_i^a \leftarrow \bar{\rho},\ \forall i \in \mathcal{A}$
6: **repeat**
7: $\quad r_i^{old} \leftarrow r_i,\ \forall i \in \mathcal{A}$
8: $\quad$ RM solves problem (P4)
9: $\quad$ **for all** $i \in \mathcal{A}$ **do**
10: $\quad\quad$ CM i solves problem (P3)
11: $\quad\quad$ **if** $\psi_i > \psi_i^{low}$ **then**
12: $\quad\quad\quad \rho_i^a \leftarrow \max\left\{\rho_i^a, \rho\right\} + \lambda \rho_i^{up}$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad \varepsilon \leftarrow \sum_{i=1}^{N} \frac{\left|r_i - r_i^{old}\right|}{r_i^{old}}$
16: **until** $\varepsilon < \bar{\varepsilon}$

Algorithm IV.2 Solution Rounding

1: sort $\mathcal{A}$ according to increasing α_i
2: $r_i \leftarrow \lceil \hat{r}_i \rceil,\ \forall i \in \mathcal{A}$
3: **for all** $j \in \mathcal{A}$ **do**
4: $\quad$ **if** $\sum_{i=1}^{N} r_i > R$ **then**
5: $\quad\quad r_j \leftarrow r_j - 1$
6: $\quad$ **end if**
7: **end for**
8: $s_i^M \leftarrow \lceil \hat{s}_i^M \rceil,\ \forall i \in \mathcal{A}$
9: $s_i^R \leftarrow \lceil \hat{s}_i^R \rceil,\ \forall i \in \mathcal{A}$
10: **for all** $j \in \mathcal{A}$ **do**
11: $\quad$ **while** $s_j^M / c_j^M + s_j^R / c_j^R > r_j$ **do**
12: $\quad\quad s_j^R \leftarrow s_j^R - 1$
13: $\quad\quad$ **if** $s_j^M / c_j^M + s_j^R / c_j^R > r_j$ **then**
14: $\quad\quad\quad s_j^M \leftarrow s_j^M - 1$
15: $\quad\quad$ **end if**
16: $\quad$ **end while**
17: **end for**

price set by the RM is not less than the unit cost for running a VM in the cluster. Further, they avoid unrealistic solutions where the RM sets a price several orders of magnitude greater than the bids without assigning any extra resources, basically ignoring penalties. The parameter enforcing this behavior is $\hat{\rho}$, defined as $\hat{\rho} \triangleq \max_i \rho_i^{up}$.

As in problem (P2) variables r_i are taken as real nonnegative. However, in this case we are not solving the continuous relaxation of the problem, as variables y_i are binary and they are treated as such. This means that model (P4) is a mixed integer nonlinear problem. In fact, even if constraint (P2d) is not part of this problem, the objective function involves the product ρr_i.

C. Iterative Approach

According to the considerations presented in Section IV-A, in this game all the CMs have strategy sets that are functions of the strategy adopted by the RM. Similarly, the strategy set of the RM depends on the bids, ρ_i^a, offered by the CMs. As previously stated, this makes the game an example of GNEP. In order to have the system converge to an equilibrium, problems (P3) and (P4) are solved iteratively until a stopping criterion is fulfilled.

Algorithm IV.1 starts assigning the initial values to all the decision variables. Then it starts a loop performing the actual iterative procedure to solve the game. First of all, the current configuration is saved in a set of auxiliary variables, since we propose to stop the computation when the total relative increment is below a given tolerance. At line 8 the RM solves its problem, setting new values for r_i and making them available to every CM. In the following nested loop, each CM solves its instance of problem (P3) and, possibly, places a higher bid for extra resources. Condition $\psi_i > \psi_i^{low}$ at line 11 checks whether CM i is rejecting any jobs: in this case, it places a higher bid trying to obtain a greater share of resources, so as to reduce the penalty-related expenditure. The parameter $\lambda \in (0, 1)$ is required to step up the bid when a CM needs more resources.

Notice that Algorithm IV.1 is meant to be solved in a distributed fashion, with the loop at line 9 executed in parallel, one iteration per CM. The algorithm execution requires that the RM sends r_i to each CM and the latter sends back the updated bids ρ_i^a. In this way, the RM does not need any sensitive knowledge on the SLAs.

D. Integer Solution Heuristic

We have previously observed that, since the continuous relaxations of the proposed models are considered for performance reasons, we need to enforce the integrality of the variables r_i, s_i^M, and s_i^R to obtain a solution useful in practice. In order to fulfill this necessity, we adopt a heuristic approach, which is presented in this section. We propose a heuristic that does not depend on how the continuous solution is obtained, hence this approach applies both to the centralized and the distributed formulations previously discussed. Algorithm IV.2 shows pseudo-code for the proposed rounding heuristic.

Constraints (P3d) are approximate formulae, hence can be relaxed when looking for nearly optimal integer solutions. Building upon this observation, in Algorithm IV.2 we propose a viable heuristic to obtain, given an optimal real solution, an integer solution that is feasible with respect to all constraints but (P3d). In the following, we denote with $\left(\hat{\mathbf{r}}, \hat{\boldsymbol{\psi}}, \hat{\mathbf{s}}^M, \hat{\mathbf{s}}^R\right)$ either the optimal solution of the continuous relaxation of our centralized resource allocation problem (P2), or the game equilibrium obtained through Algorithm IV.1.

We can easily prove that the loop starting at line 3 is enough to find a set of values of the r_i variables satisfying the following constraint:

$$\sum_{i=1}^{N} r_i \leq R. \tag{6}$$

Moreover, the nested loop at line 11 has complexity $\mathcal{O}(1)$ in the worst case.[2] Given that both the main loops are linear in N, the complexity of Algorithm IV.2 is dominated by the adopted sorting algorithm, i.e., it is $\mathcal{O}(N \log N)$. In order to shrink solution times, it is also possible to cache the sorted $\mathcal{A}$, as the α_i are fixed. Furthermore, notice that Algorithm IV.2

[2]Both proofs can be found in [13].

can be easily parallelized: in the distributed scenario, from line 8 on every CMs can separately round their variables at the same time.

V. Results

In this section, we present and examine experimental results obtained by applying the solution methods proposed in Sections III and IV. We implemented and evaluated problem (P2), named *centralized* in the following, and Algorithm IV.1 where problem (P3) is solved using the results of Proposition III.1, from now on *distributed*. Both, after completing, provide their results as input for Algorithm IV.2.

Since problem (P2) is the natural extension of the model studied and validated in [11] to a new setting with capacity constraints, the centralized algorithm is used as base case. Further, building upon the results shown in [15], we assume the validity of the underlying performance model.

The models have been implemented using the mathematical programming language AMPL [16] and solved using Knitro 9.0.1 [17]. All the analyses have been run on an Ubuntu 14.04 VM featuring 14 GB RAM hosted on an Intel Xeon E5530 2.40 GHz CPU.

A. Design of Experiments

The analyses in this section intend to be representative of real Hadoop clusters. In order to do so, the experiments have been performed considering realistic job profiles extracted from MapReduce execution logs, as in [18], where the authors report profiles obtained for batch jobs. Our instances have been randomly generate with uniform distributions over the same ranges. Accordingly, the associated deadlines are extracted within the range from 15 to 25 minutes. Further, we consider for each class a maximum concurrency level ranging from 5 to 20 users and a maximum rejection rate of 20%, hence $H_i^{low} = 0.8 H_i^{up}$.

The parameter $\bar{\rho}$ takes into account three main contributions: the energy cost related to the operation of the physical servers hosting the VMs, the overhead related to server rooms cooling, and the price of physical servers. All these contributions are normalized to obtain the time unit cost for VMs. For more details on the derivation of the value of $\bar{\rho}$, refer to [13].

Other relevant parameters are those governing the penalties for job rejection, α_i and β_i. Running the centralized model (P2) with $\psi_i^{up} = \psi_i^{low}$, so that rejection is not possible, one can determine an average job cost. Penalties can reasonably be a couple of orders of magnitude greater than the job cost, hence we take m_i, the penalty associated to the rejection of one job in class i, as 100 times the average job cost. Now it is possible to set the penalty terms appearing in the objective functions (P2a) and (P3a) equal to zero at maximum concurrency level, when no penalties are paid, and to the value computed with the above mentioned parameters at minimum concurrency. For convenience we write ψ_i^{up} and ψ_i^{low} in terms of H_i^{low} and H_i^{up}, thus getting the systems:

$$\begin{cases} \frac{\alpha_i}{H_i^{up}} - \beta_i = 0 \\ \frac{\alpha_i}{H_i^{low}} - \beta_i = m_i \left(H_i^{up} - H_i^{low} \right) \end{cases}, \quad \forall i \in \mathcal{A}, \tag{7}$$

that yield:

$$\alpha_i = m_i H_i^{up} H_i^{low}, \quad \forall i \in \mathcal{A}, \tag{8a}$$

$$\beta_i = m_i H_i^{low}, \quad \forall i \in \mathcal{A}. \tag{8b}$$

Owing to the interpretation of parameters K_i, discussed in Proposition III.1, it is easy to obtain the penalty term normalized on lacking resources, p_i. Hence, we get the formulae:

$$p_i = \frac{m_i}{K_i}, \quad \forall i \in \mathcal{A}. \tag{9}$$

In the end, we experimentally set the tolerance on the relative increment of r_i, appearing as stopping criterion in Algorithm IV.1, $\bar{\varepsilon} = 3\%$. A sensitivity analysis, omitted for space considerations, showed that the tolerance for the stopping criterion does not affect the equilibrium found with our method. Differently, we set in all the experiments $\lambda = 5\%$ as fraction of the maximum possible offer to add when raising the bid.

B. Scenario-Based Analysis

In this section we discuss some preliminary analyses performed to verify whether our formulations exhibit behaviors we intuitively expect from the modeled applications. Analyses have been run with both the previously mentioned solution approaches, considering 100 and 1,000 CMs.

In order to compare the outcomes of the solution methods, in this section we do not present results averaged on a number of random instances. Instead, we randomly generated a dataset of size 1,000, then we shrank it as needed for smaller instances, adapting capacity to the different scale.

In the MapReduce paradigm we have two main dimensions governing performance: namely, resource capacity and deadlines. Isolating each of them allows to understand how the system will react to changes in one single aspect.

1) Decreasing Capacity: To begin with, we fix the deadlines D_i and we vary the cluster capacity R. Now, starting with excess capacity, we expect to see constant running costs until the aggregate resources requirement for execution at the maximum concurrency level and the cluster capacity are comparable. If capacity keeps decreasing, at first some jobs are rejected, leading to penalties, then it falls below the minimum aggregate requirement and the problem becomes infeasible.

Notice that in this experiment we initially compute the optimal cluster size summing up all the r_i^{up} and we set $R^o = \sum_{i=1}^{N} r_i^{up}$. Then we apply our models to instances with decreasing capacity, starting from $R = 1.1 R^o$. Figure 2 shows a slight shift away from the optimal solution of the centralized problem, concentrated in proximity of the lowest feasible capacity levels.

2) Decreasing Deadlines: Alternatively, it is possible to fix the cluster capacity, R, decreasing deadlines and making them tighter. In this case, the parameters D_i act in nonlinear constraints, so we can predict an overall behavior, but the precise relation governing costs is not obvious. In the beginning, the

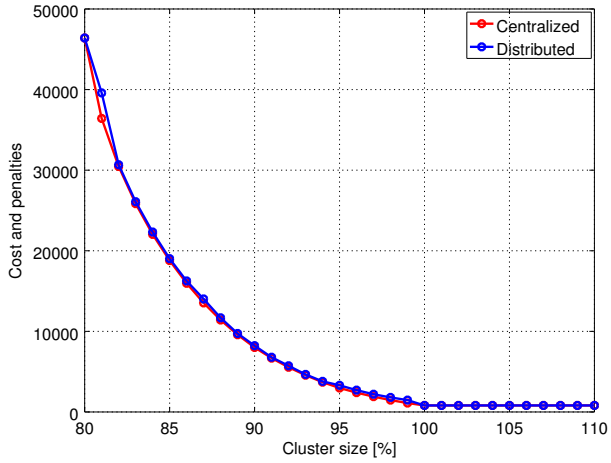

Figure 2: Decreasing capacity, 100 CMs

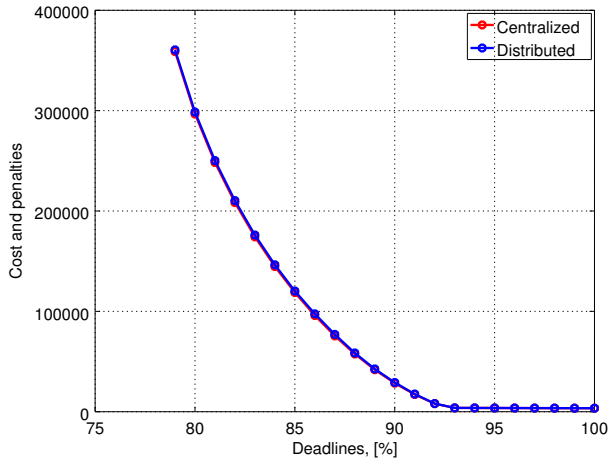

Figure 3: Decreasing deadlines, 1000 CMs

deadline reduction leads to increased running costs, due to the allocation of more VMs to reduce execution times and meet tighter constraints. Then we expect a phase where the increment of costs accelerates, as jobs start to be rejected and penalties to be paid. In the end, the deadlines become so strict to not allow even minimal operation, hence the problem turns out to be infeasible.

In this experiment, we initially set the starting values for the parameters D_i and compute the corresponding optimal capacity, $R^o = \sum_{i=1}^{N} r_i^{up}$. Then we set $R = 1.1R^o$ to start with excess capacity and gradually reduce D_i by a percentage. Figure 3 highlights a great accordance with the expected behavior.

C. Scalability Analysis

After verifying that our models satisfy some basic properties we intuitively expect, and in line with the results published in [11], we proceed with a scalability analysis. Our goal is to verify that the proposed approach allows for solving the joint admission control and capacity allocation problem at runtime, to support cluster management.

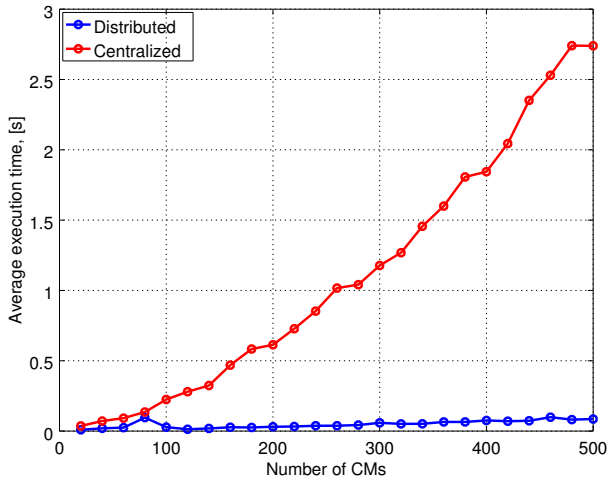

Figure 4: Solving time, scalability analysis

We progressively increase the number of CMs from 20 to 500 with step 20, performing at every step ten runs with different extractions of the randomly generated parameters, then we compute the average results. Concerning total costs and penalties, we compare objective function (P2a), obtained with the centralized, baseline approach, with the sum of all the objectives (P3a) and the aggregate cost of energy, $\sum_{i=1}^{N} \bar{\rho} r_i$, of the solutions determined with Algorithm IV.1. In this way we obtain from both the alternative solution methods the same information summarizing running costs and penalties. The solutions obtained via the alternative methods are basically equivalent, with the overall costs and penalties settling within 2% of one another.

Concerning solving times, instead, we instrumented our code to measure the timings associated to the solution algorithms. Further, whilst in the centralized approach this is enough to obtain relevant execution times, we should consider that in our testing environment Algorithm IV.1 runs on a single machine as serial code. In order to have a first approximate estimate of the actual execution times in a distributed environment, we divided the serially obtained timing by the number of CMs included in the instance at hand. This is justified as, in every iteration of Algorithm IV.1, the loop at line 9 would be executed concurrently by all the CMs at once in a distributed system. Moreover, we take into account the network-related delays by adding a term proportional to the number of iterations needed to reach convergence. Network time has been obtained by writing a micro-benchmark that sends through sockets two floats, in our system the parameters ρ and r_i coming from the RM, or ρ_i^a coming from the CMs. The benchmark was run 100 times in a 100 Mb/s network. The aggregate results are reported in Figure 4, where it is evident that the distributed approach we propose scales better than the centralized one.

VI. Related Work

Nowadays, Hadoop is widely adopted in the ICT industry, often supporting core business activities. Hence, it is of

paramount importance for users running MapReduce applications to know in advance the job completion time. Technical literature focuses on two main points of interest: on one hand, there are various works addressing methods for the estimation of job completion times and other performance metrics; on the other hand, several publications present novel approaches to resource allocation and scheduling. Two main directions have been explored: simulation-based and analytical models. The computational effort and the time spent in running simulation-based models make them hardly fit for the purposes of runtime cluster management: thus, we hereby consider only analytical models, according to the focus of our paper.

The authors of [3] propose the Automatic Resource Inference and Allocation framework for estimating the makespan of jobs in MapReduce clusters. This approach relies on information extracted from the logs of previous executions of similar jobs. Adopting scheduling techniques, the authors prove lower and upper bounds on makespans, whence they derive formulae for performance prediction that boast a relative error below 10% in the validation against measured timings.

The same research group went on in defining a performance model for Hadoop clusters [19]. They adopt job profiles for the user-supplied Map and Reduce functions, whilst the common procedures carried out by the framework are characterized via micro-benchmarks. In comparison to the measured values on a 66-node Hadoop cluster, the relative error on the predicted completion times ranges between 10% and 17%.

The model presented in [20] considers the execution time of each step in the Hadoop framework, modeling both computation and resource contention. Although this analytical model does not perform very well in reporting the real job execution time, the trend obtained when increasing the number of Map and Reduce tasks is quite reliable and can help in adjusting the number of tasks towards an optimal value.

Another performance model estimating the execution time by considering the single costs of the various phases of a MapReduce job is described in [21]. In this work, the authors go down to the very low level elements that determine the cost of single job phases, writing a 37-parameter model that provides execution times within 10% of those measured in a real cluster.

Vianna et al. [22] combine a precedence tree, which captures the relations among the different stages of the same job, and a closed queueing network (QN), to reflect resource contention within the system. The proposed method is validated through QN simulations and runs on an actual cluster, obtaining a deviation from real setup of less than 15%, but without considering multiple concurrent jobs.

The models presented before may possibly be adopted for resource allocation and admission control purposes, for instance exploiting game-theoretic or optimization techniques. However, to the best of our knowledge, in literature there exist only a few examples of papers focusing on resource allocation, even less if we consider distributed approaches. In the following, we show some works that investigate resource management and scheduling issues.

In [23], the authors propose an optimized approach to sharing clusters among MapReduce and Cloud applications. Relying on a Nash bargaining game, the authors develop modules for fair resource allocation and automatic VM migration, thus increasing resource utility and guaranteeing performance. They conduct validation on an 8-node server cluster, comparing the traditional method and the proposed hybrid environment, which outperforms the former in all the presented experiments.

Sandhom and Lai [24] address MapReduce optimization highlighting how allocating resources fairly can lead to decay in performance and efficiency: according to their proposal, users receive a budget and the system allocates resources according to each users' current spending rate. The approach is validated with experiments on a real cluster, showing improvements upon a basic, fair share strategy.

Contrasting to the presented literature, our paper couples performance prediction and resource allocation, thus building a theoretical framework capable of managing cluster resources cost-effectively, while also guaranteeing, with a certain confidence, that SLAs are met.

VII. Conclusions

In this paper we investigated the problem of resource allocation of MapReduce applications running on Hadoop clusters. In particular, we provided a scalable distributed approach for solving the joint admission control and capacity allocation problem based on a performance model available in technical literature.

Building upon the outcomes of this work, it is possible to investigate further open issues and relevant research questions. In several frameworks, such as Apache Tez and Spark, the workflows are not fixed anymore: these frameworks abstract the dependency relationships among input, output, and intermediate data with directed acyclic graphs (DAGs). An interesting development of this work is the extension of the model to the execution of complex DAGs.

In the end, we should be aware that the formulae we use to estimate performance are only approximate. A system for reliable performance prediction can greatly benefit from the coupling with a local search method based on accurate simulation techniques, such as Petri nets or QNs. In this vision, our models would provide a relevant initial guess for an iterative procedure, in order to find out the optimal configuration or, conversely, certify that an application design will respect the constraints imposed on its execution due to business considerations.

Acknowledgments

Danilo Ardagna has 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).

References

[1] J. Gantz and D. Reinsel, "The digital universe in 2020: Big Data, bigger digital shadows, and biggest growth in the Far East — United States," IDC, Tech. Rep., Feb. 2013. [Online]. Available: https://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf (visited on 01/16/2017).

[2] M. Lin, L. Zhang, A. Wierman, and J. Tan, "Joint optimization of overlapping phases in MapReduce," *SIGMETRICS Performance Evaluation Review*, vol. 41, no. 3, pp. 16–18, 2013. DOI: 10.1145/2567529.2567534.

[3] 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.

[4] P. Corcoran and A. Andrae, "Emerging trends in electricity consumption for consumer ICT," Jul. 2013. [Online]. Available: http://www.researchgate.net/publication/255923829_Emerging_Trends_in_Electricity_Consumption_for_Consumer_ICT (visited on 01/16/2017).

[5] G. Cook, "Clicking clean: A guide to building the green Internet," Greenpeace International, Tech. Rep., May 2015. [Online]. Available: http://www.greenpeace.org/usa/wp-content/uploads/legacy/Global/usa/planet3/PDFs/2015ClickingClean.pdf (visited on 01/16/2017).

[6] B. Addis, D. Ardagna, A. Capone, and G. Carello, "Energy-aware joint management of networks and Cloud infrastructures," *Computer Networks*, vol. 70, Sep. 2014.

[7] D. Ardagna, B. Panicucci, and M. Passacantando, "A game theoretic formulation of the service provisioning problem in Cloud systems," in *WWW 2011*, (Hyderabad, India), 2011, pp. 177–186.

[8] ——, "Generalized Nash equilibria for the service provisioning problem in Cloud systems," *IEEE Transactions on Services Computing*, vol. 6, no. 4, pp. 429–442, 2013.

[9] D. Ardagna, M. Ciavotta, and M. Passacantando, "Generalized Nash equilibria for the service provisioning problem in multi-Cloud systems," *IEEE Transactions on Services Computing*, Sep. 2015, ISSN: 1939-1374. DOI: 10.1109/TSC.2015.2477836, in press.

[10] M. Passacantando, D. Ardagna, and A. Savi, "Service provisioning problem in Cloud and multi-Cloud systems," *INFORMS Journal on Computing*, vol. 28, no. 2, pp. 265–277, Mar. 2016.

[11] M. Malekimajd, A. M. Rizzi, D. Ardagna, M. Ciavotta, M. Passacantando, and A. Movaghar, "Optimal capacity allocation for executing MapReduce jobs in Cloud systems," in *SYNASC*, (Timisoara, Romania), 2014, pp. 385–392.

[12] Hadoop MapReduce next generation - Capacity Scheduler, [Online]. Available: http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html (visited on 01/16/2017).

[13] E. Gianniti, D. Ardagna, M. Ciavotta, and M. Passacantando, "A game-theoretic approach for runtime capacity allocation in MapReduce," Jan. 2017. arXiv: 1701.04763 [cs.DC].

[14] F. Facchinei and C. Kanzow, "Generalized Nash equilibrium problems," *Annals of Operations Research*, vol. 175, no. 1, pp. 177–211, Mar. 2010. DOI: 10.1007/s10479-009-0653-x.

[15] 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, ISSN: 0163-5999. DOI: 10.1145/2788402.2788410.

[16] AMPL, [Online]. Available: http://www.ampl.com/ (visited on 01/16/2017).

[17] Knitro, [Online]. Available: https://www.artelys.com/en/optimization-tools/knitro (visited on 01/16/2017).

[18] A. Verma, L. Cherkasova, and R. H. Campbell, "Resource provisioning framework for MapReduce jobs with performance goals," in *Middleware*, ser. LNCS, vol. 7049, Springer Berlin Heidelberg, 2011, pp. 165–186. DOI: 10.1007/978-3-642-25821-3_9.

[19] Z. Zhang, L. Cherkasova, and B. T. Loo, "Parameterizable benchmarking framework for designing a MapReduce performance model," *Concurrency Computat.: Pract. Exper.*, 2014. DOI: 10.1002/cpe.3229.

[20] X. Yang and J. Sun, "An analytical performance model of MapReduce," in *CCIS*, IEEE, 2011.

[21] X. Lin, Z. Meng, C. Xu, and M. Wang, "A practical performance model for Hadoop MapReduce," in *International Conference on Cluster Computing Workshops*, IEEE, 2012. DOI: 10.1109/ClusterW.2012.24.

[22] E. Vianna, G. Comarela, T. Pontes, J. Almeida, V. Almeida, K. Wilkinson, H. Kumo, and U. Dayal, "Analytical performance models for MapReduce workloads," *Int J Parallel Prog*, vol. 41, pp. 495–525, 2013. DOI: 10.1007/s10766-012-0227-4.

[23] J. Zhang, F. Dong, D. Shen, and J. Luo, "Game theory based dynamic resource allocation for hybrid environment with Cloud and Big Data applications," in *International Conference on Systems, Man, and Cybernetics*, (San Diego, CA, USA), IEEE, Oct. 2014, pp. 1128–1133.

[24] T. Sandholm and K. Lai, "MapReduce optimization using regulated dynamic prioritization," in *SIGMETRICS Performance '09*, (Seattle, WA, USA), Jun. 2009, pp. 299–310.