Noname manuscript No.
(will be inserted by the editor)

An Optimization Framework for the Capacity Allocation and Admission Control of MapReduce Jobs in Cloud Systems

**M. Malekimajd · D. Ardagna ·
M. Ciavotta · E. Gianniti ·
M. Passacantando · A. M. Rizzi**

the date of receipt and acceptance should be inserted later

Abstract It is safe to claim that we live in a Big Data world. In effect, many sectors of our economy are already guided by data-driven decision processes. Big Data and Business Intelligence applications are facilitated by the MapReduce programming model while, at infrastructural layer, cloud computing provides flexible and cost effective solutions to provide on demand large clusters. Capacity allocation in such systems, meant as the problem of providing computational power to support concurrent MapReduce applications in a cost effective fashion, represents a challenge of paramount importance.

In this paper we lay the foundation for a solution implementing admission control and capacity allocation for MapReduce jobs with a priori deadline guarantees. In particular, shared Hadoop 2.x clusters supporting batch and/or interactive jobs are targeted. We formulate a linear programming model able

The simulations and numerical analyses have been performed under the Windows Azure Research Pass 2013 grant. The work of Marzieh Malekimajd has been supported by the European Commission grant no. FP7-ICT-2011-8-318484 (MODAClouds). The work of Danilo Ardagna, Michele Ciavotta, and Eugenio Gianniti has been supported by the Horizon 2020 research and innovation program under grant agreement no. 644869 (DICE). The work of Mauro Passacantando has been supported by the National Research Program no. PRIN/2015B5F27W_001 (Nonlinear and Combinatorial Aspects of Complex Networks).

M. Malekimajd
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
E-mail: malekimajd@ce.sharif.edu

D. Ardagna, M. Ciavotta, E. Gianniti, A. M. Rizzi
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Milan, Italy

M. Passacantando
Dipartimento di Informatica
Università di Pisa
Pisa, Italy

to minimize cloud resources costs and rejection penalties for the execution of jobs belonging to multiple classes with deadline guarantees. Scalability analyses demonstrated that the proposed method is able to determine the global optimal solution of the linear problem for systems including up to 10,000 classes in less than one second.

1 Introduction

Nowadays, many sectors of world economy are influenced by data-driven decision processes [30]. The availability of massive data sets and suitable distribute platforms to process them have in fact profoundly changed the way complex systems are explored and modeled, as well as the hypotheses generation process, to better grasp both the overall behavior and the component interactions. This approach is particularly important in all scenarios that hardly fit in intuitive models like natural sciences, social and engineered systems [22].

The impact of the Big Data has been epochal and its incidence has probably not yet reached its peak. Indeed, a recent McKinsey study [39] estimates that the impact of data intensive technologies on the US health care alone is worth $300 billion. The analysis also shows that the European public sector could potentially reduce expenditure of administrative activities by 15–20%, with an increase of value ranging between $223 and $446 billion [22, 39].

From the technological perspective, the MapReduce programming model is recognized to be the prominent solution for Big Data applications [35]. Its open source implementation, Hadoop, is mature and able to manage large datasets over either commodity clusters and high performance distributed topologies [56]. It has attracted interest of both industry and academia as the processing of large amounts of unstructured data became a high priority task and it overtakes the scalability level achievable with traditional data warehouses and business intelligence technologies [35].

Resource eager Big Data platforms, like Hadoop, often require very large clusters of machines to process data fast enough to support reactive business processes. For this reason, Big Data combines beautifully with the Cloud as it can provide quasi-infinite computation on a pay-per-use basis. Also, Cloud storage offers an effective and affordable solution for storing massive data set, whereas modern NoSQL databases are recognized to feature good extensibility and scalability in storing and accessing data [31]. For these reasons, many cloud providers already include in their offering MapReduce based platforms such as Microsoft HDinsight or Amazon Elastic MapReduce [6, 10]. IDC estimates that by 2020 nearly 40% of Big Data analyses will be supported by public cloud [12].

In the very beginning, MapReduce-based systems were only meant to run on dedicated clusters to support batch analyses through a FIFO scheduler [44, 45]. Over time, their mission has evolved and, nowadays, one of the main challenges is to provide tools to handle large queries, submitted concurrently by different users to a shared cluster, possibly with some guarantees on their execution time [61,62]. Solutions with strong multi-tenant capabilities already exist on the market. Hadoop in its second release, for example, has been completely redesigned to separate computational resource allocation mechanisms from application management to ensure that different applications are performed concurrently. The allocation of cluster resources, however, is still largely manual as YARN does not support performance guarantees in the resource management. This often results in a misuse of available resources, oversized clusters, and high execution costs. In this context, the ability to allocate cluster capacity optimally is a problem of particular importance yet also very challenging [49]. The problem is made even more complicated by the fact that the execution time of a MapReduce application, which also depends on the amount of resources assigned and the size of the data set to be processed, is a priori unknown and difficult to assess [36,49].

The aim of this work is to provide the theoretical foundation for the creation of a deadline-guaranteed mechanism of admission control (AC) and capacity allocation (CA) on shared Hadoop 2.x cloud clusters supporting both batch and interactive jobs. AC is an overload protection mechanism that rejects requests under peak workload conditions to prevent performance degradation. A lot of work has been done in the last decade for optimal admission control in web servers and multi-tier applications [21,24,32,33,53,54]. Recently, some work has been performed also in the Hadoop context [20,25], but to the best of our knowledge, our work is one of the first contributions that provide a joint solution for AC and CA for MapReduce jobs. In particular, we first deal with the problem of assessing the application performance by extending the work of Verma et al. [49] to the case of clusters running different applications and multiple users; we then formulate the joint AC and CA problem by means of an optimization model whose objective is to minimize both the cloud resources leasing costs and penalty costs due to rejections. In its first formulation the problem is nonlinear, following subsequent transformations a linear formulation is obtained that can be quickly solved by the solvers presently available. The scalability of the presented optimization approach is demonstrated by considering a very large set of experiments, each including up to 10,000 job classes. All the instances are solved to optimality in less than one second in the worst case.

The outcome of the model is the optimal number of cloud resources to provision a cluster managed by the YARN Capacity Scheduler [5] and the number of jobs from multiple classes that can concurrently run in it. Our work is one of the first contributions facing the problem of optimal sizing of Hadoop 2.x systems adopting the Capacity Scheduler. Further, a thorough study of the properties of the optimization model (based on our preliminary works [37,38]) leads to theoretical results that allow determining the cluster capacity in a closed form. Consequently, this result enables to establish beforehand whether a cluster will be saturated or not and, given the current cost of resources, to what extent it is cost-effective to enforce admission control policies and reject jobs.

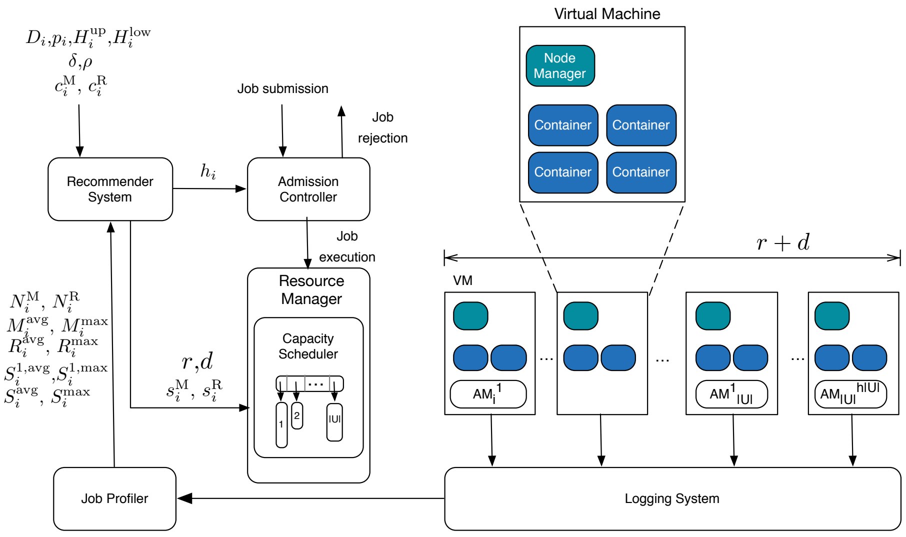


Figure 1 Hadoop 2.x reference scenario.

This paper is organized as follows. In Section 2 we describe our Hadoop 2.x reference framework and the underlying assumptions. The joint AC and CA formulation is introduced in Section 3. Experimental analyses are reported in Section 4, while Section 5 describes the related work. Conclusions are finally drawn in Section 6.

2 An Optimization Framework for the joint Capacity Allocation and Admission Control

In this section, we describe our reference system: a Hadoop 2.x cluster is governed by a resource management framework (see Figure 1) composed by a *Recommender System*, an *Admission Controller* and a *Job Profiler*.

We assume that the cloud cluster, based on HDFS, is shared among competing job classes and the system relies on the YARN Capacity Scheduler [5]. This scheduler provides (hierarchical) queues to allow sharing a large cluster while providing capacity guarantees.

YARN is the architectural core of Hadoop, which is a system for managing distributed applications. The fundamental idea of YARN is to split up the functionalities of resource management and application scheduling/monitoring into separate daemons. It consists of a central Resource Manager and per-node Node Managers, which take directions from the Resource Manager and are responsible for managing resources available on single nodes. The Resource Manager is, primarily, a pure scheduler. It arbitrates all the available cluster resources and thus helps manage the distributed applications running on the YARN system. It optimizes cluster utilization against various constraints such as capacity guarantees, fairness, and SLAs. The Resource Manager has a pluggable Scheduler that allows for different algorithms, such as capacity and fair scheduling, to be utilized as necessary. The Node Manager is the per-machine framework agent that is responsible for logically splitting the resources into *containers*, monitoring the usage and reporting to the Resource Manager/Scheduler. YARN containers are resource units (characterized by CPU and memory capacity) granted by the Resource Manager to applications via Node Managers [8]. The Resource Manager works together with Application Masters, which have the responsibility of negotiating (per application) an appropriate number of resource containers from the Scheduler, tracking their status and monitoring their progress.

In our work, we assume that jobs are submitted through an Admission Controller, meaning that some of them can be rejected [23, 59]. When a new job is submitted, the Admission Controller decides to forward the execution request to the Resource Manager or drop it. If the job is started, the Resource Manager first allocates the related Application Master, which then will receive a suitable number of YARN containers. In this configuration, the Capacity Scheduler is configured with a two-layer queue hierarchy in *work-conserving mode*[1], but speculative execution is not enabled. The first layer queue, representing 100% cluster capacity, is partitioned among $|\mathcal{U}|$ queues, one per job class. Such assumptions are introduced in order to obtain an approximate formula that allows to estimate job performance quickly and to derive a model of the joint admission control and capacity allocation problem that can be solved analytically.

The aim of the *Recommender System* is to determine the optimal number of virtual machines (VMs), either *reserved* or *on demand* (see Section 3.1), the system is to use and the concurrency levels for each class, minimizing the overall costs due to cloud resources and job rejections. Based on the configuration of YARN, the allocated VMs will host the containers needed for running the accepted jobs. Formally, the proposed recommender solves a joint AC and CA problem which is the core of this paper contribution, formulated in the next section, at fixed intervals of time reacting to changes in the workload composition.

3 Optimization Model

In this section, we first summarize the main assumptions (Section 3.1) and some grounding results we achieved in determining bounds and approximate formulae for estimating MapReduce jobs execution time in shared Hadoop 2.x cluster (Section 3.2). We then formulate the joint AC and CA problem for the execution of MapReduce jobs in cloud environments and show that it is equivalent to a linear programming problem (Section 3.3). Afterwards, in

[1] A scheduler is defined to be *work conserving* if it never lets a processor lie idle while there are runnable tasks in the system, i.e., Application Masters in a queue can borrow containers from other empty queues.

Profile Parameters	
$\mathcal{U}$	Set of job classes
c_i^{M}	Maximum number of Map containers that can be hosted in a VM of class i
c_i^{R}	Maximum number of Reduce containers that can be hosted in a VM of class i
ξ_i^{M}	CPU time requirement for the Map phase of job class i
ξ_i^{R}	CPU time requirement for the Reduce phase of job class i
ξ_i^{0}	Constant time term that depends on Map, Shuffle, and Reduce phases of class i jobs

Table 1 Optimization model: profile parameters.

SLA Parameters	
p_i	Penalty for rejecting jobs from class i
D_i	Maximum duration for class i jobs
H_i^{up}	Upper bound on the number of class i jobs to be executed concurrently
H_i^{low}	Lower bound on the number of class i jobs to be executed concurrently

Table 2 Optimization model: SLA parameters.

Section 3.4 we point out some important properties of the optimal solutions of the problem, which are the starting point to derive, in Section 3.5, the closed-form optimal solution of a system including two job classes.

3.1 Model Assumptions

In this paper we consider pure MapReduce job (we are planning to support Tez [4] Directed Acyclic Graphs (DAGs) in our future work) consisting of two main data manipulation phases, namely Map and Reduce; each phase performs a user-defined function on input data. During the Map phase, the input data is divided into input splits for analysis by Map tasks running in parallel across the Hadoop cluster. The Reduce phase uses results from Map tasks as input to a set of parallel Reduce tasks. Between the Map processing and the Reduce processing, a shuffle step sorts all map output values with the same key into a single reduce input.

Jobs are characterized by an *execution profile*, defined as the set of features used to estimate the execution time, and a *deadline*, that is an upper bound on the execution time. Jobs with the similar profiles and deadlines define a class $i \in \mathcal{U}$. As in [16,49], the profile encompasses the number of Map and Reduce tasks (namely, N_i^{M} and N_i^{R}), and the average and maximum duration of Map, Reduce, first Shuffle and typical Shuffle[2] phases (denoted by M_i^{avg}, M_i^{max}, R_i^{avg}, R_i^{max}, $S_i^{1,\mathrm{avg}}$, $S_i^{1,\mathrm{max}}$, S_i^{avg} and S_i^{max}). Execution profiles can be obtained, for instance, through the analysis of logs of previous runs [29,43,47] and we assume that the execution time of the individual Map and Reduce

[2] During the Shuffle phase data from the mapper tasks are moved to the nodes where the reducer tasks will run. As in [49] we distinguish between the first and the typical Shuffle, since they are characterized by significantly different performance.

stages is independent of the system load as a first approximation. Note that estimates on how the total Map, Reduce, and Shuffle duration depend on the data set size can also be modeled [49,51]. Job deadlines will be denoted by D_i.

Each class i is also identified by a certain concurrency level (denoted by h_i), i.e., the number of class i jobs allowed to execute concurrently. H_i^{up} denotes a design-time prediction on the number of job per class i to be executed, hence we have $h_i \leq H_i^{\text{up}}$. Furthermore, in order to avoid job starvation, we also impose h_i to be greater than a given lower bound H_i^{low}; p_i indicates the penalty due per class i job rejection.

For the sake of simplicity, we assume that all the YARN containers, as well as all the supporting computational nodes (the VMs that host the containers), are homogeneous with respect to memory and CPU. This setup is frequently adopted by cloud providers [1,11] since it limits the occurrence of stragglers (i.e., slower nodes that, due to the strict synchronization between the Map and Reduce stages, limit job performance). Furthermore, as a first approximation, we assume that the full container capacity can be assigned to either Map or Reduce tasks[3] (YARN assigns resources dynamically, hence there is not a static partitioning of slots as it was the case for Hadoop 1.x) and job profiling is independent of the cluster size (this assumption holds as a first approximation if network is not the bottleneck [2,7,18]).

The last assumption concerns the cost model adopted by our reference system. We assume that our reference system implementation is hosted in a cloud environment that provides homogeneous VMs. Inspired by the Amazon EC2 pricing model [6], we consider two possible computational resources, referred to as *reserved* and *on demand* instances, respectively. Both types are billed on an hourly basis but, while the former can be booked with an upfront payment that guarantees availability and lower hourly price, the latter can be requested without long-term commitments at a relatively more expensive fee. The hourly prices of reserved and on demand VMs are denoted by ρ and δ, respectively, with $\delta > \rho$, while $\bar{r}$ refers to the number of VMs that have been reserved with a long-term contract, that is the maximum number of reserved VMs that can be leased at discounted price ρ.

3.2 Approximate Formulae for MapReduce Execution Time

Although MapReduce job performance metrics can be evaluated, for example, by relying on simulations [14,27], there is a fundamental trade-off between the

[3] Note that in Hadoop 1.X, each node resources can be partitioned between slots assigned to Map tasks and slots assigned to Reduce tasks. In Hadoop 2.x, the resource capacity configured for each container is suitable to both Map and Reduce tasks and cannot be partitioned anymore [7]. The maximum number of concurrent mappers and reducers (the slot count) is calculated by YARN based on administrator settings [2]. A node is eligible to run a task when its available memory and CPU can satisfy the task resource requirement. With our hypothesis above, we assume that the configuration settings is such that whatever combination of Map and Reduce tasks can be executed within a container, no vCPU remains idle because of a wrong setting of these parameters.

accuracy of the models and the time required to run them. Given the need to compute capacity allocation at scale (Hadoop clusters nowadays run thousands of jobs a day [48]), the high complexity of simulating even small-scale instances of MapReduce jobs has prevented us from exploiting such results here. That is why we have opted for a fast approximate approach (yet in line with the accuracy requested for capacity planning activities [34]).

In [16] we demonstrated that, if we denote s_i^{M} the number of Map containers and s_i^{R} the number of Reduce containers devoted to the execution of class i jobs, then both an upper bound and an approximate formula for the job execution time can be expressed as:

$$T_i = \frac{\xi_i^{\mathrm{M}} h_i}{s_i^{\mathrm{M}}} + \frac{\xi_i^{\mathrm{R}} h_i}{s_i^{\mathrm{R}}} + \xi_i^0, \tag{1}$$

where ξ_i^{M}, ξ_i^{R} and ξ_i^0 are positive constants depending on the class execution profile, the cluster resources and on the nature of the formula (upper bound or average time estimate). It can be noted that the Shuffle phase does not appear on its own in formula (1). This is due to the fact that the bulk of the related processing happens during the synchronization between Map and Reduce, without any dependency on the resource allocation, but only on the dataset size: this effect is taken into account with an additive term in ξ_i^0. Another smaller fraction of the Shuffle-related service CPU time requirement is observed as preliminary work in Reduce tasks, then it is part of ξ_i^{R}. The expressions for ξ_i^{M}, ξ_i^{R}, and ξ_i^0 can be found in Appendix A.

In the case that equation (1) represents an upper bound for the execution time, the resource allocation for a given class will be conservative and D_i can be considered hard deadlines. Vice versa, D_i have to be considered soft deadlines. The experimental results we presented in [16] demonstrated that such approximate formulae are accurate (with respect to simulation) and the difference between the simulated job execution time and our estimation ranges between 5 and 10%, while the gap for upper bounds is between 11 and 19%.

Let us denote c_i^{M} and c_i^{R} the maximum number of Map and Reduce containers that can be hosted in each VM, respectively, i.e., each instance supports either c_i^{M} Map or c_i^{R} Reduce concurrent tasks for each job class i, depending on how YARN is configured with respect to vCPUs and memory [3]. As a consequence, in order to provide s_i^{M} and s_i^{R} Map and Reduce containers to a certain class, the number of VMs to be provisioned is bounded by $s_i^{\mathrm{M}}/c_i^{\mathrm{M}} + s_i^{\mathrm{R}}/c_i^{\mathrm{R}}$.

3.3 Problem Formulation

The aim of the joint AC and CA problem solved by the *Recommender System* is to minimize the overall execution cost meeting, at the same time, the job deadlines. The execution cost includes the VM leasing costs and rejection penalties. Given p_i the penalty cost for the rejection of one class i job, the

Provider Parameters	
δ	Cost of on demand VMs
ρ	Cost of reserved VMs
$\bar{r}$	Number of available reserved VMs

Table 3 Optimization model: provider parameters.

Decision Variables	
s_i^{M}	Number of YARN containers to be allocated to class i for executing Map tasks
s_i^{R}	Number of YARN containers to be allocated to class i for executing Reduce tasks
h_i	Number of class i jobs to be executed concurrently
r	Number of reserved VMs to be provisioned
d	Number of on demand VMs to be provisioned

Table 4 Optimization model: decision variables.

overall execution cost can be calculated as

$$\delta d + \rho r + \sum_{i \in \mathcal{U}} p_i \left(H_i^{\mathrm{up}} - h_i \right), \tag{2}$$

where we do not consider any term associated to execution times under the assumption that the deadlines D_i are all shorter than an hour. This means that all the accepted jobs will finish within an hour from the provisioning of the VMs and, therefore, the workload will complete in the one-hour time slot billed by the cloud provider. Hence, the cost we incur per VM is exactly the hourly price.

The decision variables are d, r, h_i, s_i^{M} and s_i^{R} for any $i \in \mathcal{U}$, i.e., the system has to provide an indication on the overall number of reserved and on demand VMs, plus the concurrency level and, as a by-product, the number of Map and Reduce containers for every class i. The notation adopted in this paper is summarized in Tables 1, 2, 3 and 4.

The optimization problem can then be defined as follows:

$$\min_{d, r, \mathbf{h}, \mathbf{s}^{\mathrm{M}}, \mathbf{s}^{\mathrm{R}}} \; \delta\, d + \rho\, r - \sum_{i \in \mathcal{U}} p_i\, h_i \tag{P1a}$$

subject to:

$$\frac{\xi_i^{\mathrm{M}}\, h_i}{s_i^{\mathrm{M}}} + \frac{\xi_i^{\mathrm{R}}\, h_i}{s_i^{\mathrm{R}}} + \zeta_i^{0,D} \le 0, \quad \forall i \in \mathcal{U}, \tag{P1b}$$

$$r \le \bar{r}, \tag{P1c}$$

$$\sum_{i \in \mathcal{U}} \left(\frac{s_i^{\mathrm{M}}}{c_i^{\mathrm{M}}} + \frac{s_i^{\mathrm{R}}}{c_i^{\mathrm{R}}} \right) \le r + d, \tag{P1d}$$

$$H_i^{\mathrm{low}} \le h_i \le H_i^{\mathrm{up}}, \quad \forall i \in \mathcal{U}, \tag{P1e}$$

$$r \ge 0, \tag{P1f}$$

$$d \ge 0, \tag{P1g}$$

$$s_i^{\mathrm{M}} \ge 0, \quad \forall i \in \mathcal{U}, \tag{P1h}$$

$$s_i^{\mathrm{R}} \ge 0, \quad \forall i \in \mathcal{U}. \tag{P1i}$$

The model aims at minimizing the cluster operational costs without exceeding either hard or soft deadlines. The objective function includes VM provisioning costs and potential penalties for job rejection. Note that, with respect to formula (2), the term $\sum_{i \in \mathcal{U}} p_i\, H_i^{\mathrm{up}}$ has been removed as it is independent of the decision variables. Constraints (P1b) are derived from equation (1) by imposing that each job class must be completed before the respective deadline, that is requiring that $T_i \le D_i$, where D_i is the maximum duration allowed for to class i jobs (deadline). By denoting $\zeta_i^{0,D}$ the difference $\zeta_i^{0,D} = \xi_i^0 - D_i < 0$, constraints (P1b) are easily derived. Constraint (P1c) bounds the total number of reserved VMs to provision, whereas (P1d) guarantees that the number of allocated VMs is sufficient to support the submitted jobs and meet their deadlines. Indeed, the term $s_i^{\mathrm{M}}/c_i^{\mathrm{M}} + s_i^{\mathrm{R}}/c_i^{\mathrm{R}}$ quantifies the number of VMs required for class i by dividing the number of YARN containers required by the job (s_i^{M} and s_i^{R}) by the capacity of each computational node (i.e., the number of containers that can be hosted, according to the way the scheduler is configured). Constraints (P1e) bound the concurrency level for each class, while constraints (P1f)–(P1i) specify the definition domain of all the variables involved.

We remark that, although the variables participating in the formulation represent indivisible entities (number of VMs), we did not impose the integrality, as this would have greatly increased the solution times to the point of being useless for the envisioned scenario. This approximation is widely used in the literature (see, e.g., [15,58]) since relaxed variables can be rounded to the closest integer at the expense of a generally small increment of the overall cost, especially for large-scale MapReduce clusters running tens or hundreds of VMs.

Formulation (P1) features a linear objective function. However, constraints (P1b) are nonconvex (the proof is reported in [37]) and difficult to address even by the most effective nonlinear solvers. To remedy this inconvenience, the problem can be reformulated using a new set of decision variables $\Psi_i = 1/h_i$ for all $i \in \mathcal{U}$ in lieu of h_i. The formulation resulting from this transformation is shown below:

$$\min_{d,r,\boldsymbol{\Psi},\mathbf{s}^{\mathrm{M}},\mathbf{s}^{\mathrm{R}}} \quad \delta\, d + \rho\, r - \sum_{i\in\mathcal{U}} \frac{p_i}{\Psi_i} \tag{P2a}$$

subject to:

$$\frac{\xi_i^{\mathrm{M}}}{s_i^{\mathrm{M}}\Psi_i} + \frac{\xi_i^{\mathrm{R}}}{s_i^{\mathrm{R}}\Psi_i} + \zeta_i^{0,D} \le 0, \quad \forall i \in \mathcal{U}, \tag{P2b}$$

$$r \le \bar{r}, \tag{P2c}$$

$$\sum_{i\in\mathcal{U}} \left(\frac{s_i^{\mathrm{M}}}{c_i^{\mathrm{M}}} + \frac{s_i^{\mathrm{R}}}{c_i^{\mathrm{R}}} \right) \le r + d, \tag{P2d}$$

$$\Psi_i^{\mathrm{low}} \le \Psi_i \le \Psi_i^{\mathrm{up}}, \quad \forall i \in \mathcal{U}, \tag{P2e}$$

$$r \ge 0, \tag{P2f}$$

$$d \ge 0, \tag{P2g}$$

$$s_i^{\mathrm{M}} \ge 0, \quad \forall i \in \mathcal{U}, \tag{P2h}$$

$$s_i^{\mathrm{R}} \ge 0, \quad \forall i \in \mathcal{U}. \tag{P2i}$$

where $\Psi_i^{\mathrm{low}} = 1/H_i^{\mathrm{up}}$ and $\Psi_i^{\mathrm{up}} = 1/H_i^{\mathrm{low}}$.

As a consequence of this transformation, constraints (P2b), corresponding to the constraints (P1b), are convex. The convexity is condition underlying several important results. Indeed, Theorem 1 below provides in closed-form the number s_i^{M} and s_i^{R} of YARN containers to be allocated to each class i for executing Map and Reduce tasks, respectively. The importance of this result is double: it allows computing the optimal YARN capacity scheduler weights α_i in terms of s_i^{M} and s_i^{R}, and reformulating problem (P2) as an equivalent linear programming problem (see Theorem 2) which can be solved very efficiently by commercial solvers. Finally, Theorem 3 is an important and innovative theoretical result that describes some properties of the optimal solutions of the linear programming model: 1) the execution cost of one job, 2) establishing a priori if the execution of a job is profitable or has to be rejected, 3) under which conditions on demand VMs will be used.

To the best of our knowledge, our work is the first contribution providing a linear programming model that solves jointly the admission control and capacity allocation problem providing formulae that allow estimating the number of VMs that are needed to support MapReduce jobs with a priori performance guarantees.

Theorem 1 *In any optimal solution of problem* (P2)*, constraints* (P2b) *hold as equalities. Moreover, for any $i \in \mathcal{U}$, the number s_i^M and s_i^R of YARN containers to be allocated to support the execution of h_i jobs of class i are given by the following formulae:*

$$s_i^M = -\frac{h_i}{\zeta_i^{0,D}}\left(\sqrt{\frac{\xi_i^M\,\xi_i^R\,c_i^M}{c_i^R}} + \xi_i^M\right), \tag{3}$$

$$s_i^R = -\frac{h_i}{\zeta_i^{0,D}}\left(\sqrt{\frac{\xi_i^M\,\xi_i^R\,c_i^R}{c_i^M}} + \xi_i^R\right). \tag{4}$$

Proof See Appendix B.

Theorem 1 allows transforming (P2) into an equivalent linear programming formulation. In fact, equations (3) and (4) express variables s_i^{M} and s_i^{R} as linear functions of h_i, thus reducing the number of variables. In order to present the model in a simplified form, we introduce the following constants:

$$\gamma_i^1 = -\frac{1}{\zeta_i^{0,D}\,c_i^{\mathrm{M}}}\left(\sqrt{\frac{\xi_i^{\mathrm{M}}\,\xi_i^{\mathrm{R}}\,c_i^{\mathrm{M}}}{c_i^{\mathrm{R}}}} + \xi_i^{\mathrm{M}}\right), \tag{5}$$

$$\gamma_i^2 = -\frac{1}{\zeta_i^{0,D}\,c_i^{\mathrm{R}}}\left(\sqrt{\frac{\xi_i^{\mathrm{M}}\,\xi_i^{\mathrm{R}}\,c_i^{\mathrm{R}}}{c_i^{\mathrm{M}}}} + \xi_i^{\mathrm{R}}\right). \tag{6}$$

Remark 1 Since h_i is the number of concurrent jobs of class i supported by the allocated VMs, the terms s_i^{M}/h_i and s_i^{R}/h_i represent the number of Map and Reduce tasks of class i, respectively, that the cluster must uphold to guarantee the execution of one single class i job. Therefore, it follows from equations (3)–(6) that

$$\gamma_i^1 = \frac{s_i^{\mathrm{M}}}{h_i\,c_i^{\mathrm{M}}} \qquad \text{and} \qquad \gamma_i^2 = \frac{s_i^{\mathrm{R}}}{h_i\,c_i^{\mathrm{R}}},$$

i.e., γ_i^1 and γ_i^2 indicate the number of VMs required to support the Map and Reduce task of one class i job, respectively.

Theorem 2 *Problem* (P2) *is equivalent to the following linear programming problem* (P3)*:*

$$\min_{d,r,\mathbf{h}} \ \delta\,d + \rho\,r - \sum_{i\in\mathcal{U}} p_i\,h_i \tag{P3a}$$

subject to:

$$\sum_{i\in\mathcal{U}} \gamma_i\,h_i \le r + d, \tag{P3b}$$

$$0 \le r \le \bar{r}, \tag{P3c}$$

$$d \ge 0, \tag{P3d}$$

$$H_i^{low} \le h_i \le H_i^{up}, \quad \forall\ i\ \in \mathcal{U}, \tag{P3e}$$

where the decision variables are d, r and h_i for any $i \in \mathcal{U}$, and parameters $\gamma_i = \gamma_i^1 + \gamma_i^2$.

Proof See Appendix B.

3.4 Properties of optimal solutions

The Karush-Kuhn-Tucker (KKT) conditions corresponding to problem (P3) allow proving that any optimal solution of (P3) has the following important properties.

Theorem 3 *If* (d^*, r^*, h^*) *is an optimal solution of problem* (P3)*, then the following statements hold:*

1. $r^* > 0$*, i.e., reserved instances are always used.*
2. $\sum_{i \in \mathcal{U}} \gamma_i \, h_i^* = r^* + d^*$*, i.e.,* γ_i *can be considered a computing capacity conversion ratio that allows translating class* i *concurrency levels into resource requirements.*
3. *If* $p_i > \gamma_i \, \delta$*, then* $h_i^* = H_i^{up}$*, i.e., class* i *jobs are never rejected.*
4. *If* $p_i < \gamma_i \, \rho$*, then* $h_i^* = H_i^{low}$*, i.e., class* i *concurrency level is set to the lower bound.*
5. *If* $\bar{r} > \sum_{i \in \mathcal{U}} \gamma_i \, H_i^{up}$*, then* $d^* = 0$*: given property* (2) *interpretation, if the total capacity requirement can be satisfied through reserved instances, on demand VMs are never used.*
6. *If* $\bar{r} < \sum_{i \in \mathcal{U}} \gamma_i \, H_i^{low}$*, then* $r^* = \bar{r}$ *and* $d^* > 0$*, i.e., for property* (2)*, if the minimum job requirement exceeds reserved instance capacity, then on demand VMs are needed.*

Proof See Appendix B.

Property (1) is obvious, since reserved instances are the cheapest ones. Property (2) leads to the most important theoretical result of this paper. Indeed, the γ_i parameters can be interpreted as capacity conversion ratios that make it possible to directly estimate VM requirements depending on class i concurrency level. In light of such consideration, also properties (3) and (4) become intuitive. The product $\gamma_i \, \delta$ represents the unit cost for the execution of one class i job using on demand instances. If $\gamma_i \, \delta$ is lower than the penalty cost, then class i jobs will always be executed, i.e., $h_i = H_i^{\mathrm{up}}$. Conversely, if $\gamma_i \, \rho$, i.e., the class i job unit cost using reserved instances, is greater than the penalty, then class i jobs are always rejected, yielding $h_i = H_i^{\mathrm{low}}$.

Finally, properties (5) and (6) relate the overall minimum and maximum capacity requirements ($\sum_{i \in \mathcal{U}} \gamma_i \, H_i^{\mathrm{low}}$ and $\sum_{i \in \mathcal{U}} \gamma_i \, H_i^{\mathrm{up}}$, respectively) to reserved capacity and allow establishing a priori whether or not on demand VMs will be used and if the reserved VMs cluster will be saturated.

3.5 A Two-Class Case Study

In order to gain further insight into the properties of the optima in our setting, let us focus on the two-class case, i.e., $|\mathcal{U}| = 2$. This particular case is important as it is possible to obtain analytically a closed-form formulation of the optimal solution (d^*, r^*, h_1^*, h_2^*) of problem (P3).

The analysis shows that the formulae characterizing the optimal solution (d^*, r^*, h_1^*, h_2^*) of problem (P3) vary depending solely on the value of $\bar{r}$, ρ and δ.

Table 5 reports the optimal solution for all the possible cases (see the proof in Appendix B wherein we assume, without loss of generality, that $p_1/\gamma_1 < p_2/\gamma_2$). Rows compare the ratio between penalties (p_i) and number of VMs per single job (γ_i) compared to on demand (δ) and reserved costs (ρ). Six are the possible cases, numbered from 1 to 6. The columns relate the reserved computational capacity ($\bar{r}$) with the minimum and maximum number of resources ($\gamma_i H_i^{\text{low}}$ and $\gamma_i H_i^{\text{up}}$, respectively) to be allocated to run full load jobs. The four scenarios impacting the optimal solution are denoted by letter *a* to *d*.

In particular, rows 1 and 6 represent the most extreme cases where high (low) hourly machine leasing cost entail minimum (maximum) concurrency levels. It is interesting to note that in the case of high leasing costs and low number of reserved VMs (row 1, column a), the system is forced to allocate all the available reserved machines, i.e., $r^* = \bar{r}$. However, since the capacity requirement is not satisfied, the consequent imbalance must be corrected by renting a suitable number of on demand machines ($d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - \bar{r}$). This, obviously, does not occur when reserved resources are sufficient to fulfill the requirements (cases b to d). Similar considerations apply for the the case of low leasing costs (row 6). In this case, however, due to the high concurrency levels allowed, the depletion of reserved VMs is a more common scenario (columns a to c).

Rows from 2 to 5 present a more complex solution structure wherein peculiar and nontrivial situations emerge. It is the case, for instance, of cell 2b (row 2, column b) in which the optimal concurrency level for class 1 (h_1^*) is set to the minimum possible value (as $p_1/\gamma_1 < \rho < \delta$) whereas h_2^* depends on the residual capacity of the cluster.

More details on the optimal solution for the two-class case are discussed in Section 4.2.

4 Experimental Results

In this section we: (i) present the setup of our experiments and evaluate the scalability of the joint AC/CA problem solution, (ii) provide a geometrical description for the two-class case properties, (iii) investigate how different (P3) problem settings impact on the leasing costs of the provided cloud cluster, (iv) evaluate the gap between the job execution time and the deadline through simulation and experiments on Amazon EC2.

The following section presents the scalability study, whereas Section 4.2 analyzes the two-class results investigating how our problem solution behaves in different settings, while the analysis of Section 4.3 provides some insights on the impact of the most important parameters of our model on the cluster costs. Finally, Section 4.4 reports the results of the deadline guarantees study.

	a)	*b)*	*c)*	*d)*
	$\bar{r} < \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$	$\gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} < \bar{r} < \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$	$\gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} < \bar{r} < \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$	$\bar{r} > \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$
1) $\frac{p_1}{\gamma_1} < \frac{p_2}{\gamma_2} < \rho < \delta$	$r^* = \bar{r}$ $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - \bar{r}$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{low}}$	$r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$ $d^* = 0$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{low}}$		
2) $\frac{p_1}{\gamma_1} < \rho < \frac{p_2}{\gamma_2} < \delta$	$r^* = \bar{r}$ $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - \bar{r}$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{low}}$	$r^* = \bar{r}$ $d^* = 0$ $h_1^* = H_1^{\text{low}}$ $h_2^* = (\bar{r} - \gamma_1 H_1^{\text{low}})/\gamma_2$	$r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$ $d^* = 0$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{up}}$	
3) $\frac{p_1}{\gamma_1} < \rho < \delta < \frac{p_2}{\gamma_2}$	$r^* = \bar{r}$ $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} - \bar{r}$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{up}}$		$r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$ $d^* = 0$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{up}}$	
4) $\rho < \frac{p_1}{\gamma_1} < \frac{p_2}{\gamma_2} < \delta$	$r^* = \bar{r}$ $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - \bar{r}$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{low}}$	$r^* = \bar{r}$ $d^* = 0$ $h_1^* = H_1^{\text{low}}$ $h_2^* = (\bar{r} - \gamma_1 H_1^{\text{low}})/\gamma_2$	$r^* = \bar{r}$ $d^* = 0$ $h_1^* = (\bar{r} - \gamma_2 H_2^{\text{up}})/\gamma_1$ $h_2^* = H_2^{\text{up}}$	$r^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$ $d^* = 0$ $h_1^* = H_1^{\text{up}}$ $h_2^* = H_2^{\text{up}}$
5) $\rho < \frac{p_1}{\gamma_1} < \delta < \frac{p_2}{\gamma_2}$	$r^* = \bar{r}$ $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} - \bar{r}$ $h_1^* = H_1^{\text{low}}$ $h_2^* = H_2^{\text{up}}$		$r^* = \bar{r}$ $d^* = 0$ $h_1^* = (\bar{r} - \gamma_2 H_2^{\text{up}})/\gamma_1$ $h_2^* = H_2^{\text{up}}$	$r^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$ $d^* = 0$ $h_1^* = H_1^{\text{up}}$ $h_2^* = H_2^{\text{up}}$
6) $\rho < \delta < \frac{p_1}{\gamma_1} < \frac{p_2}{\gamma_2}$	$r^* = \bar{r}$ $d^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}} - \bar{r}$ $h_1^* = H_1^{\text{up}}$ $h_2^* = H_2^{\text{up}}$			$r^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$ $d^* = 0$ $h_1^* = H_1^{\text{up}}$ $h_2^* = H_2^{\text{up}}$

Table 5 Closed-form optimal solution of problem (P3) for the two-class case.

4.1 Scalability Analysis

In this section, we report the scalability results of our optimization solution. The analyses have been performed on a VirtualBox virtual machine based on Ubuntu Server 12.04 running on an Intel Xeon Nehalem dual socket quad-core system with 32 GB of RAM. Optimal solutions of (P3) were obtained by running CPLEX 12.0 where we also restricted the decision variables d, r, h_i to be integer, i.e., we considered the Integer Linear Programming (ILP) version of (P3). For completeness, we also show the solving times in the case where the admission control mechanism is disabled. Note that in this scenario the variables h_i in (P3) are known, hence the problem can be further simplified.

For all the analyses presented in this work, instances have been randomly generated as follows. We used uniform distributions within the ranges reported in Table 6. These ranges have been chosen according to values observed in real systems and MapReduce job logs. To determine a reasonable value for penalties, we first evaluated the minimum cost associated with the execution a single job, C_i, by setting $H_i^{\text{up}} = H_i^{\text{low}} = 1$ and solving problem (P3), thus disabling the admission control mechanism. Then, we determined the penalty value for rejections as $p_i = 10\, C_i$, following [13]. We varied H_i^{up} in the range $[10, 30]$, and we set $H_i^{\text{low}} = 0.9\, H_i^{\text{up}}$. Moreover, as in [50], we assumed that deadlines are uniformly distributed in the range $[25, 45]$ minutes. The experiments are

Job Profile	
N_i^{M}	[70, 700]
N_i^{R}	[32, 64]
$M_i^{\max}$ [s]	[16, 120]
S_i^{avg} [s]	[24, 120]
$S_i^{\max}$ [s]	[30, 150]
$R_i^{\max}$ [s]	[15, 75]
$S_i^{1,\max}$ [s]	[10, 30]
$c_i^{\mathrm{M}}, c_i^{\mathrm{R}}$	[1, 4]
D_i [s]	[600, 1200]

Class Population	
$H_i^{\mathrm{up}}/H_i^{\mathrm{low}}$	[10, 30]

Rejection Penalty	
p_i [¢]	[250, 2500]

Cloud Instance Price	
ρ [¢]	[5, 20]
δ [¢]	[5, 40]

Table 6 Cluster characteristics and Job Profiles.

performed considering different class sizes, varying the cardinality of the set $\mathcal{U}$ between 100 and 10,000 with step 100. Finally, for a given instance size, each experiment is executed ten times to mitigate the impact of possible disturbs in the system.

The results reported in Figure 2 show that the time required to determine the global optimal solution of the ILP problem can reach up to 1967 ms in the worst case, but takes less than a second in every other case: a time significantly lower than the one required to start a large Hadoop cluster in a public cloud. Note that a system with 10,000 classes is at least one order of magnitude larger than scenarios of practical interest[4]. Hence, we argue that our recommendation system can be used in practice for the configuration of real Hadoop clusters. Figure 2 also shows the measures related to the solution of (P3) when considering only capacity allocation. Since they are generally smaller by orders of magnitude than solving the ILP, Figure 3 details only these values. Similarly, there is a clear linear dependency on the number of classes considered in each instance, yet the maximum solution time is 23 ms.

4.2 Two-class Case Analysis

The purpose of this section is to give the reader a clear vision of the equations reported in Table 5. To do this, we report a three-dimensional graphical representation of the behavior of the decision variables h_1 (Figure 4) and r^* (Figure 5) relative to the half-plane defined by the axes p_1/γ_1 and p_2/γ_2, with $p_1/\gamma_1 < p_2/\gamma_2$. The variables h_2 and d^* follow a similar pattern and, therefore, they have not been reported here. Instead, we decided to present a small set of scenarios that cover the cases a, b, c, and d (i.e., the columns of Table 5) under several conditions (the rows of the same table). It is worth noticing that, geometrically, such conditions split the definition domain in several areas corresponding to possible different values of h_1 and r^*, which have a staircase shape.

[4] For example, the TPC-DS benchmark, designed to be representative of real data warehouse systems, includes 99 queries that, in the worst case, can be modeled as individual job classes.

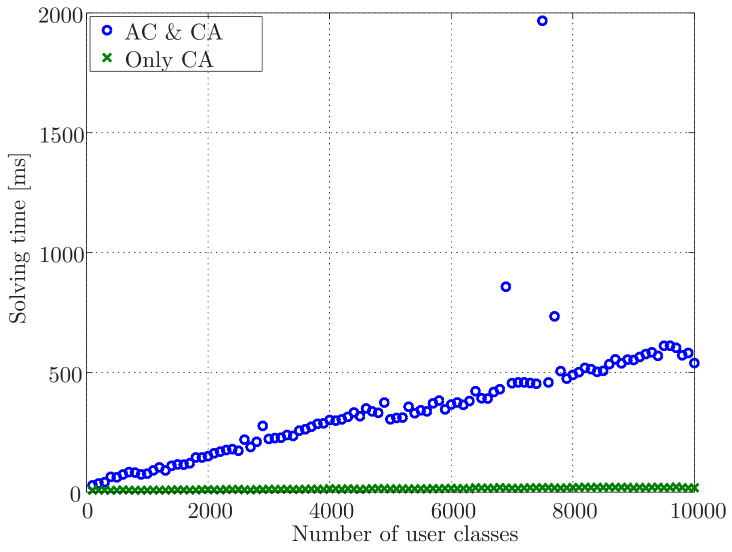


Figure 2 Solving times with and without admission control.

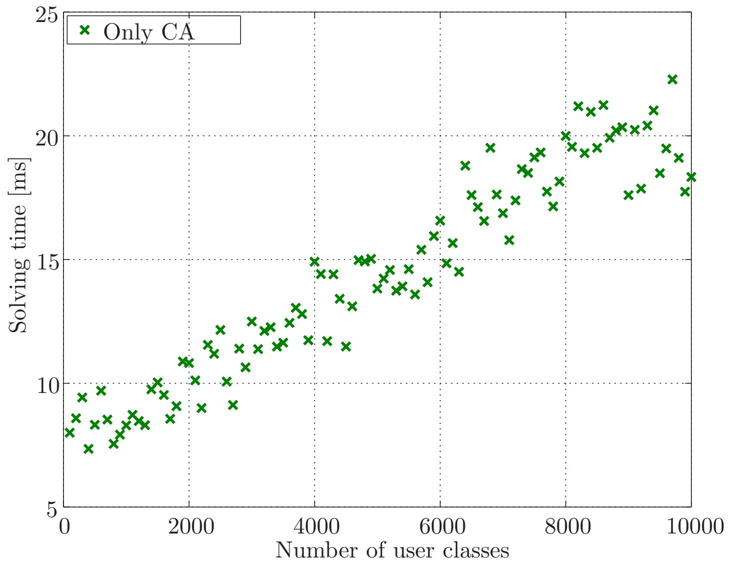
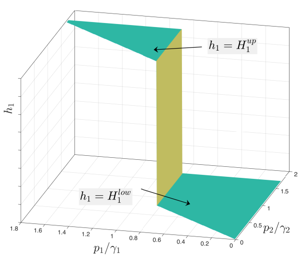
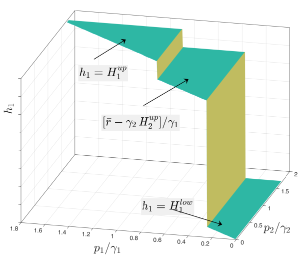


Figure 3 Solving times without admission control.

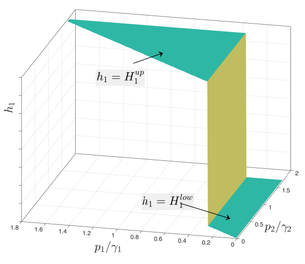
In more detail, Figure 4a represents h_1 under the condition $\bar{r} < \gamma_1\, H_1^{\mathrm{low}} + \gamma_2\, H_2^{\mathrm{up}}$ (a and b of Table 5) and in this case h_1 can assume only two values: H_1^{up}, if $\delta < p_1/\gamma_1$, and H_1^{low}, otherwise. A similar pattern can be identified in Figure 4c, where the discriminant condition is $\rho < p_1\gamma_1 < p_2\gamma_2 < \delta$. Condition c (i.e., $\gamma_1\, H_1^{\mathrm{low}} + \gamma_2\, H_2^{\mathrm{up}} < \bar{r} < \gamma_1\, H_1^{\mathrm{up}} + \gamma_2\, H_2^{\mathrm{up}}$), instead, is represented



(a) h_1: conditions a and b



(b) h_1: condition c



(c) h_1: condition d

Figure 4 Two-class use case: h_1 distribution for different values of p_1/γ_1 and p_2/γ_2 when $p_1/\gamma_1 < p_2/\gamma_2$.

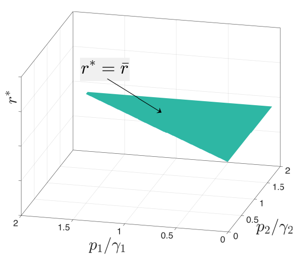
in Figure 4b, where h_1 shows three possible values, namely H_1^{low} (conditions 1, 2, 3), $(\bar{r} - \gamma_2 H_2^{\text{up}})/\gamma_1$ (conditions 4 and 5) and H_1^{up} (condition 6).

As regards the distribution of the optimal number r^* of reserved VMs, we can see that, if condition a holds, the system exploits all the VMs available (Figure 5a), whereas in the remaining cases two or three values of r^* are possible. In particular, the lowest possible value is, in the latter case, equal to $\gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$ and the highest value can be either $\bar{r}$ (conditions b and c) or $\gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$ (condition d).

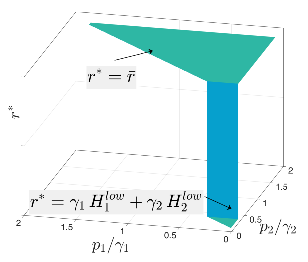
4.3 Case Studies

In this section, we investigate how different settings for problem (P3) affect the cloud cluster costs. In particular, we analyze three case studies to address the following research questions: (1) Is it better to consider a shared cluster or to devote a dedicated cluster to individual job classes? (2) What is the effect of concurrency on the cluster costs? (3) What is the impact on the cost of more strict deadlines? Along the same lines, is there a linear relation between the costs and job deadlines?

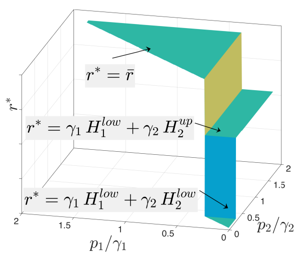
The studied instances have been generated as explained in Section 4.1. Moreover, to ease the interpretation of outcomes we excluded reserved VMs and assumed a single pricing model. Additionally, we exploited the results



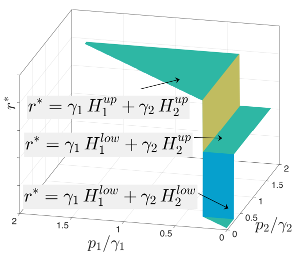
(a) r^*: condition a



(b) r^*: condition b



(c) r^*: condition c



(d) r^*: condition d

Figure 5 Two-class use case: r^* distribution for different values of p_1/γ_1 and $p_2\gamma_2$ when $p_1\gamma_1 < p_2/\gamma_2$.

gathered for each research question to confirm the applicability of Theorem 3, devised for a relaxed linear model, also in case of integer variables.

4.3.1 Effect of Cluster Sharing

In this case study, we want to examine the effect of cluster resource sharing. In particular, we considered two scenarios. The first one is our baseline, taking into account the presented (P3) formulation. The second one considers the same workload, in terms of job profiles, deadlines, etc., but $|\mathcal{U}|$ (P3) single-class problems are solved independently, which is equivalent to assuming a dedicated cluster devoted to each job class. To perform the comparisons we consider different numbers of classes. We varied the cardinality of the set $\mathcal{U}$ between 20 and 1,000 with step 20, randomly generating ten instances per cardinality value.

For each instance we calculate two values: the first one is the objective function of the baseline scenario, to which we refer as *dependent objective function*; the second one, which we call *independent objective function*, is evaluated by summing up the $|\mathcal{U}|$ objective functions of the individual problems. The comparison is performed by considering the ratio between the dependent and independent objective function. Figure 6 illustrates the average of these ratios for different numbers of job classes $|\mathcal{U}|$. Overall, the multi-class formulation allows saving on deployments: on average, we have a 0.48% variation of

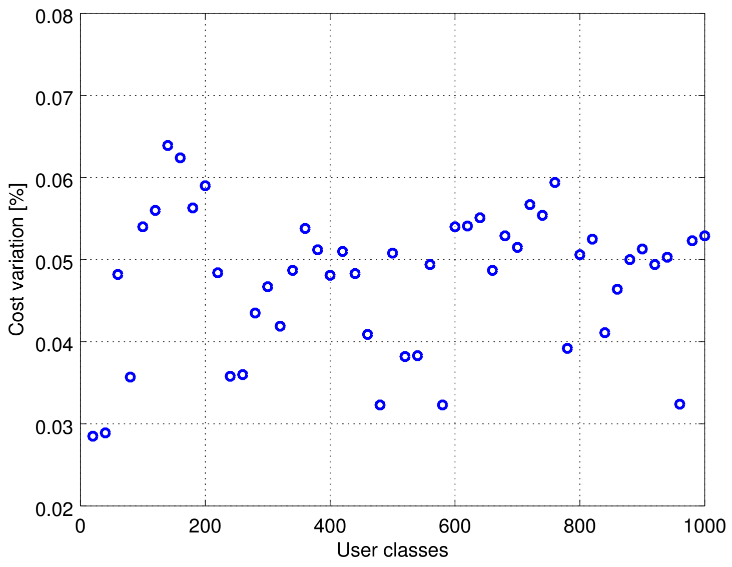


Figure 6 Cost variation between shared and dedicated clusters.

the total cluster costs. We can conclude that, thanks to cloud elasticity, the adoption of shared or dedicated clusters substantially leads to the same costs. Note, however, that on one side shared clusters can further benefit from a single HDFS instance (e.g., better disk performance and node load balancing, one copy of the data sets instead of several replicas). On the other hand, since a larger number of jobs introduces higher resource contention, according to the results we achieved in [26], there is an upper bound in terms of number of job/classes beyond which shared clusters performance degrade and dedicated clusters should be favored. Unfortunately, these effects cannot be captured by the simplified cost and performance models we adopt in this paper (we refer the reader to [26]).

Finally, note that this study confirms the validity of Theorem 3 when integer variables are considered. In fact, property (2) guarantees that, for continuous variables, the two considered scenarios are equivalent in terms of costs; for instance, if we consider two classes, the costs associated with the shared cluster are $\sum_{i \in \{1,2\}} \delta\, \gamma_i\, h_i^* = \delta\, \gamma_1\, h_1^* + \delta\, \gamma_2\, h_2^*$, which is equivalent to the sum of costs of two dedicated clusters. Enforcing integrality does not seem to disrupt this property, based on the reported results.

4.3.2 Effect of the Concurrency Level

In this case study the analysis of the effect of the concurrency level on the costs associated to the execution of one single job is performed. In the experiment, we assumed there is only one job class in the cluster. The concurrency level h_i has been varied from 10 to 30 and, for each value, 10 instances of problem (P3) have been randomly generated. For each instance, we disabled the admission

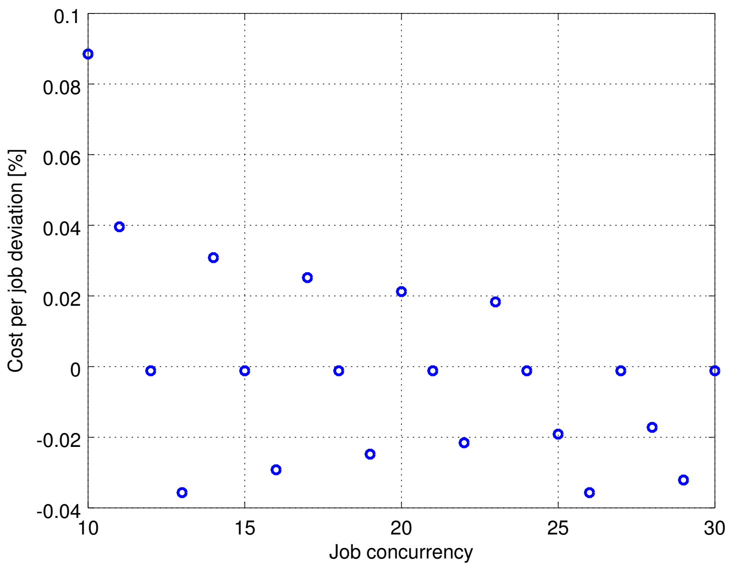


Figure 7 Effect of concurrency level on single job cost.

control mechanism by imposing $H_i^{\text{low}} = H_i^{\text{up}}$ and we solved the optimization problem. We calculated the cost of one single job for every instance by dividing the objective function by the concurrency level.

Figure 7 shows how the per-job cost varies with different concurrency levels for a representative example. Per-job costs are not shown with their absolute values, but with relative deviations with respect to the overall average per-job cost. The analysis demonstrates that the cost variation for different concurrency levels is negligible, i.e., the different concurrency levels lead to a less than 0.1% variation of the cost of one job. Hence, in a cloud setting, elasticity allows to obtain constant per-job execution costs independently from the number of jobs in a class. This result is in line with the property (2) of Theorem 3.

4.3.3 Effect of Tightening the Deadlines

Here we want to examine the relationship between costs and deadlines. In particular, we analyzed the effect of reducing the deadlines on cluster costs. The number of classes $|\mathcal{U}|$ has been varied between 20 and 1,000 and for each problem instance size several random instances have been generated, as described in Section 4.1. For each instance, the deadlines of every class have iteratively tightened to observe how this is reflected on the costs. At each step, we decreased the deadlines by 5% of the initial value. The reduction process continues until the problem becomes unfeasible. After each reduction we calculated the increased cost ratio, i.e., the ratio between the objective function for the problem with the new deadlines and the objective function of the problem with the initial deadlines, thus evaluating both the higher

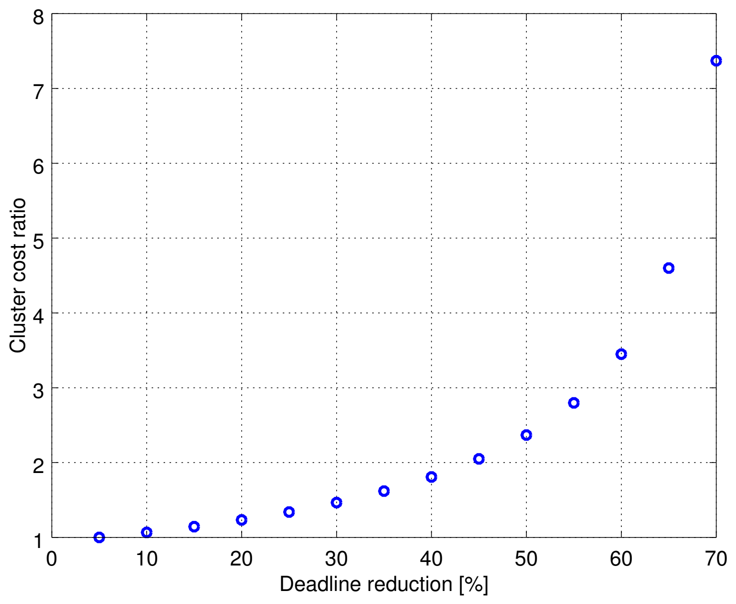


Figure 8 Effect of reducing deadlines on cluster cost.

operating costs and the effect of penalties. Figure 8 illustrates the trend of the increased cost ratio for a representative instance with 20 classes: the effect on costs of decreasing deadlines is not linear and the cost to pay for reducing the deadlines by 60% is more than three times higher than the baseline. Note that the cost ratio curve is approximated, according to Theorem 3, by the function $\alpha / (\beta - x)$ where x is the percentage reduction on deadlines, $\alpha = 71$ and $\beta = 80$ with maximum deviation equal to 2.90%.

4.4 Performance Model Validation

In order to validate the analytical performance model proposed in equation (1), we performed experiments both with YARN SLS simulator and on clusters deployed in Amazon EC2.

4.4.1 Validation through Simulation

In this section, we evaluate our approach by computing the percentage deviation between the simulated execution time and the deadline D_i imposed on job execution. Indeed, according to the properties discussed in Section 3.3, we expect the makespan of each run to approach the deadline imposed as problem parameter.

Simulations are based on our extension of the YARN SLS tool, the official Hadoop simulator we reported in [38].

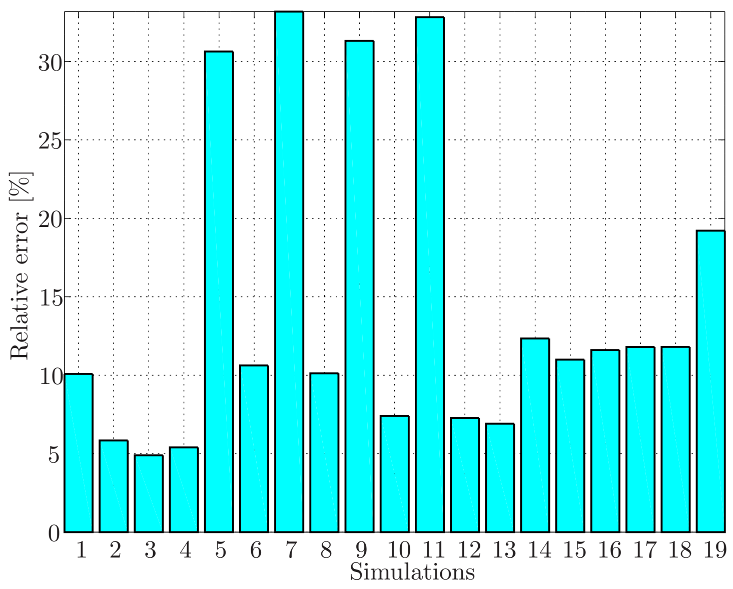


Figure 9 Prediction accuracy vs. simulations.

We considered different test configurations with three job classes with parameters randomly generated as discussed in previous sections. These scenarios represent light load conditions that correspond to the worst case for the evaluation of our bounds. Indeed, under light load conditions the probability that any job class is temporarily idle can be significant and the Capacity scheduler would assign the idle class slots to other classes to boost their performance. Vice versa, under heavy loads our upper bounds become tighter.

The results are summarized in Figure 9. On average, the absolute gap of the simulated execution times with respect to the deadline is 14.44%.

4.4.2 Validation on a Real System

Our model has been validated also on a Hadoop 2.6.0 cluster hosted in the cloud and operating on Amazon AWS. All the VMs were m4.xlarge instances with 4 vCPUs, 16 GB RAM, 500 GB SSD, and running Ubuntu Server 12.04 as operating system. The VMs ran under the same placement group, thus they were interconnected via a 1 Gbps network. A master node was dedicated to run the Ambari server to deploy and manage other Hadoop services on the nodes of the cluster. Moreover, 30 slave nodes were available as DataNodes and computational nodes. Three among these slave nodes hosted the server instances, e.g., YARN ResourceManager, HDFS NameNode, etc.

The dataset used for testing has been generated using the TPC-DS benchmark[5] data generator, creating at a scale factor of 250 GB several text files

[5] `http://www.tpc.org/tpcds/`

```
select inv_item_sk , inv_warehouse_sk
from inventory
where inv_quantity_on_hand > 10
group by inv_item_sk , inv_warehouse_sk
having sum(inv_quantity_on_hand) > 20
limit 100
```

(a) Q1

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

(b) Q2

```
select cs_item_sk , avg(cs_quantity) as aq
from catalog_sales
where cs_quantity > 2
group by cs_item_sk
```

(c) Q3

```
select inv_warehouse_sk , sum(inv_quantity_on_hand)
from inventory
group by inv_warehouse_sk
having sum(inv_quantity_on_hand) > 5
limit 100
```

(d) Q4

Figure 10 Hive MapReduce queries.

directly used as external tables by Hive. We chose the TPC-DS benchmark as it is the industry standard for benchmarking data warehouses.

Subsequently, we performed experiments on four *ad hoc* Hive queries, inspired by the ones that are part of TPC-DS, but modified so as to yield exactly one MapReduce job: see Figure 10 for the listings. The profiling phase was conducted on a dedicated cluster, extracting average task durations from 50 runs of each query. The numbers of Map and Reduce tasks varied, respectively, in the ranges $(65, 524)$ and $(5, 384)$.

We ran our model according to the experimental configuration proposed in Section 4.4.1, considering as costs the actual Amazon pricing for m4.xlarge VMs and allowing up to 30 nodes. The instances considered three classes: the first was Q1, then a class for Q2 and Q3, since they showed similar behavior, and a third one with Q4. Afterwards, we configured the YARN Capacity

H_1^{up}	H_2^{up}	H_3^{up}	d_1	d_2	d_3	D [s]	MAPE [%]
5	0	5	26	0	4	86	28.17
0	10	10	0	17	1	1,000	33.88
5	20	0	1	25	0	1,300	20.47

Table 7 Prediction accuracy on real system.

Scheduler according to the results of the optimization process and launched the mentioned Hive queries concurrently, with a 10 s think time. Each query was run at least 37 times. Table 7 reports the results of three different experiments: each row lists the concurrency levels per class, the unique deadline imposed on all of them, the assigned resources, and the mean average percentage error (MAPE) of that instance, where we compare the measured response times to the deadline. In these experiments we disabled the admission control mechanism by imposing $H_i^{\text{low}} = H_i^{\text{up}}$ for any i. The average relative error observed on the real system is 27.51%, higher than in the case of simulations, but still acceptable for the prediction of response times and in line with the accuracy expected with capacity planning activities [34].

5 Related Work

Capacity allocation and optimal scheduling of Hadoop jobs have been broadly analyzed in literature. The work in [29] presents *Starfish*, a self-tuning system for analytics on Hadoop. Starfish is able to profile a Hadoop job at runtime and automatically derive an optimized configuration which can be used for subsequent jobs. The same tool has been successful employed to solve cluster sizing problems [28].

The work in [47] addresses resource provisioning optimization, trying to minimize the cost associated with the execution of an application. The work presents a cost model that depends on the amount of input data and on the characteristics of the considered application. The model parameters are estimated through a regression-based analysis. A framework to support application profiling and estimation of the application duration on heterogeneous hardware is proposed in [51].

Authors in [17] propose a different approach, based on closed queuing networks. The work considers also contention and parallelism on compute nodes to evaluate the completion time of a MapReduce job, but it only addresses Map phase execution time. The work in [52] presents a similar solution, but its evaluation only considers clusters executing a single job. Reduce-intensive jobs are addressed in [46], modeling the Map phase as an M/G/1 queue and the Reduce phase as a multi-server queue. [36] models the execution of Map tasks through a tandem queue with overlapping phases. It also provides a very efficient run time scheduler, solving the joint optimization of Map and Shuffle phases, comparable with an off-line optimal solution.

The work in [19] introduces a novel mean field analysis approach to devise very fast approximate methods able to predict the performance of Big Data systems. Authors in [42] handle MapReduce jobs with deadlines. The work overcomes Hadoop schedulers inability to properly execute jobs within deadlines by adopting multiprocessor scheduling policies. The evaluation shows that using two versions of the earliest deadline first heuristic the approach can outperform the standard Hadoop schedulers. The work in [59] proposes a similar solution, able to partition a cluster among Hadoop jobs and traditional Web systems.

Parallax [40] is a progress estimator able to predict the completion time of parallel queries, executed as MapReduce workloads. The estimator is implemented on Pig and it has been evaluated with PigMix benchmark. *Para-Timer* [41], an extension of Parallax, supporting predictions also on queries expressed as a DAG of MapReduce jobs. The work in [60] analyzes the performance of MapReduce jobs on Hadoop cloud-based clusters both homogeneous and heterogeneous. The work provides a framework for minimizing infrastructural costs based on simulation, closely related to the approach followed in this paper. However, such framework does not address admission control and can optimize a single kind of workload, corresponding to a single class. For what concerns admission control, several solutions were proposed in the last decade for optimal management of web servers and multi-tier applications [21, 24, 32, 33, 53, 54]. The basic idea is to predict the value of a specific performance metric/server queue and if such value grows above a certain threshold, the admission controller rejects all new sessions favoring the service of requests from already admitted sessions. While a lot of work has been done for such class of systems in the Big Data research area [63] proposed an overload protection mechanisms for Database-as-a-Service (DaaS) environments. The work developed a profit-aware admission control policy which relies on nonlinear regression to predict the probability for a query to meet its performance requirement, and then decides whether the query should be admitted to the database system or not. To the best of our knowledge, even if admission control has been introduced recently in Big Data frameworks [9, 57], our work is one of the few contributions [20, 25] that faces the optimal admission control for MapReduce jobs. The work in [20] proposes a control theoretic approach to perform scaling and admission control of MapReduce cluster, in order to reduce utilization costs while preserving guarantees on both performance and availability. Authors in [25] present a learning based opportunistic algorithm admitting MapReduce able to meet deadlines in more than 80% of cases. The incoming jobs labeling is performed using a Naive Bayes Classifier, and then, from the admissible jobs, a job expected to maximize service provider utility is picked.

The work in [55] addresses optimization of mixed interactive and batch workloads running on system on a chip with heterogeneous cores. The ARIA framework [49] tries to minimize the amount of resources (containers) to be allocated to Map and Reduce tasks in order to meet a user-defined soft deadline, thus reducing the costs due to resource over-provisioning. This work is the closest to our contribution; it targets clusters with single-job classes on top of a FIFO scheduler. It determines from a compact MapReduce job profile a lower bound, an upper bound and an estimation of the execution time. The performance model, improved in [62], has been extensively validated through simulation and on a 66-node Hadoop cluster. The work in [61] optimizes a workload specified as a set of MapReduce DAGs with a global deadline or budget constraint on heterogeneous clusters, also considering the presence of faulty nodes.

This paper extends our preliminary work [37,38] by providing additional theoretical results that enable to obtain the number of VMs needed per class and per job through closed formulae and to determine a priori if the execution of a job is profitable.

6 Conclusions and Future Work

In this paper, we presented an optimization model for minimizing the execution costs of heterogeneous jobs in shared Hadoop cloud clusters. The focus of our work is on the development of an optimization framework able to exploit the YARN hierarchical architecture. The problem we faced is to determine the optimal number of cluster nodes and identify the execution and rejection rates for the execution of MapReduce workloads with a priori deadline guarantees. Our method is one of the first contributions for the optimal sizing of Hadoop 2.x systems based on the Capacity Scheduler that determine solutions in a closed form. Overall, the proposed method enables to establish beforehand whether a cluster will be saturated or not and, given the current cost of on demand resources, to what extent it is cost-effective to enforce admission control policies and job rejection. Our proposed solution has been validated by a large set of experiments. Results have shown that our method is able to determine the global minimum solutions for systems including up to 10,000 job classes in less than one second on average. In our research agenda, we plan to extend the optimization model to consider also the sizing of Spark jobs and to cope with different cloud pricing models.

References

1. Amazon elastic mapreduce.
2. Apache Hadoop YARN: Avoiding 6 Time-Consuming “Gotchas”. http://blog.cloudera.com/blog/2014/04/apache-hadoop-yarn-avoiding-6-time-consuming-gotchas/.
3. Apache Hadoop YARN: Avoiding 6 Time-Consuming "Gotchas". http://blog.cloudera.com/blog/2014/04/apache-hadoop-yarn-avoiding-6-time-consuming-gotchas/.
4. Apache tez.
5. Capacity scheduler. http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.
6. Elastic compute cloud (ec2). http://aws.amazon.com/ec2.
7. Getting MapReduce 2 Up to Speed. http://blog.cloudera.com/blog/2014/02/getting-mapreduce-2-up-to-speed/.
8. Hadoop Yarn.

9. Impala admission control. https://www.cloudera.com/documentation/enterprise/5-12-x/topics/impala_admission.html.
10. Microsoft Azure. http://azure.microsoft.com/en-us/services/hdinsight/.
11. Microsoft hdinsight.
12. The digital universe in 2020. http://idcdocserv.com/1414.
13. J. Anselmi, D. Ardagna, and M. Passacantando. Generalized Nash equilibria for SaaS/-PaaS Clouds. *European Journal of Operational Research*, 236(1):326–339, 2014.
14. D. Ardagna, S. Bernardi, E. Gianniti, S. K. Aliabadi, D. Perez-Palacin, and J. I. Requeno. Modeling performance of hadoop applications: A journey from queueing networks to stochastic well formed nets. In J. Carretero, J. G. Blas, R. K. L. Ko, P. Mueller, and K. Nakano, editors, *Algorithms and Architectures for Parallel Processing - 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings*, volume 10048 of *Lecture Notes in Computer Science*, pages 599–613. Springer, 2016.
15. D. Ardagna, B. Panicucci, and M. Passacantando. Generalized Nash Equilibria for the Service Provisioning Problem in Cloud Systems. *IEEE Trans. Services Computing*, 6(4):429–442, 2013.
16. D. Ardagna and M. S. Squillante. Special issue on performance and resource management in big data applications. *SIGMETRICS Performance Evaluation Review*, 42(4):2, 2015.
17. S. Bardhan and D. A. Menascé. Queuing network models to predict the completion time of the map phase of mapreduce jobs. In *Int. CMG Conference*, 2012.
18. A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri. Exploiting mean field analysis to model performances of big data architectures. *Future Generation Comp. Syst.*, 37:203–211, 2014.
19. A. Castiglione, M. Gribaudo, M. Iacono, and F. Palmieri. Exploiting mean field analysis to model performances of big data architectures. *Future Generation Computer Systems*, 37(0):203 – 211, 2014.
20. S. Cerf, M. Berekmeri, B. Robu, N. Marchand, and S. Bouchenak. Towards control of mapreduce performance and availability. In *Fast Abstract in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.
21. R. Chang, editor. *2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24-29, 2012*. IEEE Computer Society, 2012.
22. C. P. Chen and C.-Y. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275(0):314 – 347, 2014.
23. C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservation-based scheduling: If you're late don't blame us! In *SoCC*, 2014.
24. C. Delimitrou, N. Bambos, and C. Kozyrakis. Qos-aware admission control in heterogeneous datacenters. In J. O. Kephart, C. Pu, and X. Zhu, editors, *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013*, pages 291–296. USENIX Association, 2013.
25. J. Dhok, N. Maheshwari, and V. Varma. Learning based opportunistic admission control algorithm for mapreduce as a service. In S. Padmanabhuni, S. K. Aggarwal, and U. Bellur, editors, *Proceeding of the 3rd Annual India Software Engineering Conference, ISEC 2010, Mysore, India, February 25-27, 2010*, pages 153–160. ACM, 2010.
26. E. Gianniti, A. M. Rizzi, E. Barbierato, M. Gribaudo, and D. Ardagna. Fluid petri nets for the performance evaluation of mapreduce and spark applications. *SIGMETRICS Performance Evaluation Review*, 44(4):23–36, 2017.
27. A. Gómez, J. Merseguer, E. D. Nitto, and D. A. Tamburri. Towards a UML profile for data intensive applications. In D. Ardagna, G. Casale, A. van Hoorn, and F. Willnecker, editors, *Proceedings of the 2nd International Workshop on Quality-Aware DevOps, QUDOS@ISSTA 2016, Saarbrücken, Germany, July 21, 2016*, pages 18–23. ACM, 2016.
28. H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *SOCC*, 2011.
29. H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, 2011.
30. 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*, 57(7):86–94, July 2014.

31. K. Kambatla, G. Kollias, V. Kumar, and A. Grama. Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 74(7):2561 – 2573, 2014.
32. H. Khazaei, J. V. Misic, V. B. Misic, and S. Rashwand. Analysis of a pool management scheme for cloud computing centers. *IEEE Trans. Parallel Distrib. Syst.*, 24(5):849–861, 2013.
33. K. Konstanteli, T. Cucinotta, K. Psychas, and T. A. Varvarigou. Admission control for elastic cloud services. In Chang [21], pages 41–48.
34. E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative system performance - computer system analysis using queueing network models*. Prentice Hall, 1984.
35. K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel data processing with mapreduce: A survey. *SIGMOD Rec.*, 40(4):11–20, Jan. 2012.
36. M. Lin, L. Zhang, A. Wierman, and J. Tan. Joint optimization of overlapping phases in MapReduce. *SIGMETRICS Performance Evaluation Review*, 41(3):16–18, 2013.
37. M. Malekimajd, A. M. Rizzi, D. Ardagna, M. Ciavotta, M. Passacantando, and A. Movaghar. Optimal Capacity Allocation for executing Map Reduce Jobs in Cloud Systems Politecnico di Milano Technical Report n. 2014.11. http://home.deib.polimi.it/ardagna/MapReduceTechReport2014-11.pdf.
38. 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 F. Winkler, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie, editors, *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, Timisoara, Romania, September 22-25, 2014*, pages 385–392. IEEE Computer Society, 2014.
39. J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute, 2012.
40. K. Morton, M. Balazinska, and D. Grossman. Paratimer: A progress indicator for mapreduce dags. In *SIGMOD*, 2010.
41. K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of mapreduce pipelines. In *ICDE*, 2010.
42. L. T. X. Phan, Z. Zhang, Q. Zheng, B. T. Loo, and I. Lee. An empirical analysis of scheduling techniques for real-time cloud-based data processing. In *SOCA*, 2011.
43. N. Poggi, D. Carrera, A. Call, S. Mendoza, Y. Becerra, J. Torres, E. Ayguadé, F. Gagliardi, J. Labarta, R. Reinauer, N. Vujic, D. Green, and J. A. Blakeley. Aloja: A systematic study of hadoop deployment variables to enable automated characterization of cost-effectiveness. In *BigData Conference*, pages 905–913, 2014.
44. J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguad, M. Steinder, and I. Whalley. Performance-driven task co-scheduling for mapreduce environments. In *NOMS*, 2010.
45. B. T. Rao and L. S. S. Reddy. Survey on improved scheduling in hadoop mapreduce in cloud environments. *CoRR*, abs/1207.0780, 2012.
46. J. Tan, Y. Wang, W. Yu, and L. Zhang. Non-work-conserving effects in mapreduce: diffusion limit and criticality. In *SIGMETRICS*, 2014.
47. F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In *CLOUD*, 2011.
48. 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 G. M. Lohman, editor, *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 5:1–5:16. ACM, 2013.
49. A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *ICAC*, 2011.
50. A. Verma, L. Cherkasova, and R. H. Campbell. Resource provisioning framework for mapreduce jobs with performance goals. In *Middleware*, 2011.
51. A. Verma, L. Cherkasova, and R. H. Campbell. Profiling and evaluating hardware choices for mapreduce environments: An application-aware approach. *Performance Evaluation*, 79:328–344, 2014.

52. E. Vianna, G. Comarela, T. Pontes, J. M. Almeida, V. A. F. Almeida, K. Wilkinson, H. A. Kuno, and U. Dayal. Analytical performance models for mapreduce workloads. *International Journal of Parallel Programming*, 41(4):495–525, 2013.
53. L. Wu, S. K. Garg, and R. Buyya. Sla-based admission control for a software-as-a-service provider in cloud computing environments. *J. Comput. Syst. Sci.*, 78(5):1280–1299, 2012.
54. P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacigümüs. Activesla: a profit-oriented admission control framework for database-as-a-service providers. In J. S. Chase and A. E. Abbadi, editors, *ACM Symposium on Cloud Computing in conjunction with SOSP 2011, SOCC '11, Cascais, Portugal, October 26-28, 2011*, page 15. ACM, 2011.
55. F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni. Heterogeneous cores for mapreduce processing: Opportunity or challenge? In *NOMS*, 2014.
56. F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni. Optimizing power and performance trade-offs of mapreduce job processing with heterogeneous multi-core processors. In *CLOUD*, 2014.
57. Y. Yao, J. Lin, J. Wang, N. Mi, and B. Sheng. Admission control in YARN clusters based on dynamic resource reservation. In R. Badonnel, J. Xiao, S. Ata, F. D. Turck, V. Groza, and C. R. P. dos Santos, editors, *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11-15 May, 2015*, pages 838–841. IEEE, 2015.
58. Q. Zhang, Q. Zhu, M. F. Zhani, and R. Boutaba. Dynamic service placement in geographically distributed clouds. *ICDCS*, 2012.
59. W. Zhang, S. Rajasekaran, S. Duan, T. Wood, and M. Zhu. Minimizing interference and maximizing progress for hadoop virtual machines. *SIGMETRICS Performance Evaluation Review*, 42(4):62–71, 2015.
60. Z. Zhang, L. Cherkasova, and B. T. Loo. Exploiting cloud heterogeneity for optimized cost/performance mapreduce processing. In *CloudDP*, 2014.
61. Z. Zhang, L. Cherkasova, and B. T. Loo. Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing. *SIGMETRICS Performance Evaluation Review*, 42(4):38–50, 2015.
62. Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo. Automated Profiling and Resource Management of Pig Programs for Meeting Service Level Objectives. In *ICAC*, 2012.
63. P. P. Xiong, Y. Chi, S. Zhu,J. Tatemura, C. Pu, H. Hacigümüs. ActiveSLA: a profit-oriented admission control framework for database-as-a-service providers. In *SOCC*, 2011.

A Makespan Bounds

In the following we report the results we presented in [16] providing an approximated formula for the estimation of MapReduce jobs. We consider a MapReduce system with up to s_i^{M} and s_i^{R} containers devoted for the Map and Reduce phase using the Capacity Scheduler.

Following the results in [49], the lower and upper bounds on the duration of the entire Map stage can be estimated as follows:

$$T_i^{\mathrm{M,low}} = \frac{N_i^{\mathrm{M}} M_i^{\mathrm{avg}}}{s_i^{\mathrm{M}}} h_i$$

$$T_i^{\mathrm{M,up}} = \frac{N_i^{\mathrm{M}} M_i^{\mathrm{avg}} - 2M_i^{\mathrm{max}}}{s_i^{\mathrm{M}}} h_i + 2M_i^{\mathrm{max}}$$

where M_i^{avg}, M_i^{max}, R_i^{avg}, R_i^{max}, $S_i^{1,\mathrm{avg}}$, $S_i^{1,\mathrm{max}}$, S_i^{avg}, and S_i^{max} denote the average and maximum duration of Map, Reduce, first Shuffle and typical Shuffle phases, respectively, while N_i^{M} and N_i^{R} are the number of Map and Reduce tasks (see Section 2). According to the results discussed in [49], we distinguish the non-overlapping portion of the first shuffle and the task durations in the typical shuffle. In the following bounds for the shuffle stage, this consideration affects the formula for $T_i^{\mathrm{S,low}}$, where we subtract one wave:

$$T_i^{\mathrm{S,low}} = \left(\frac{N_i^{\mathrm{R}}}{s_i^{\mathrm{R}}} h_i - 1\right) S_i^{\mathrm{avg}},$$
$$T_i^{\mathrm{S,up}} = \frac{N_i^{\mathrm{R}} S_i^{\mathrm{avg}} - 2S_i^{\max}}{s_i^{\mathrm{R}}} h_i + 2S_i^{\max}.$$

$S_i^{1,\mathrm{avg}}$ and $S_i^{1,\max}$, the average and maximum execution time of the first shuffle phase, are estimated directly from the execution profile of class i.

Summing up all parts we get:

$$T_i^{\mathrm{low}} = T_i^{\mathrm{M,low}} + S_i^{1,\mathrm{avg}} + T_i^{\mathrm{S,low}} + T_i^{\mathrm{R,up}},$$
$$T_i^{\mathrm{up}} = T_i^{\mathrm{M,up}} + S_i^{1,\max} + T_i^{\mathrm{S,up}} + T_i^{\mathrm{R,low}}.$$

T_i^{low} and T_i^{up} represent,respectively, an optimistic and a pessimistic prediction of class i job completion time. We also define $T_i^{\mathrm{avg}} = \left(T_i^{\mathrm{up}} + T_i^{\mathrm{low}}\right)/2$. Hence, the execution time of a class i job is at least:

$$T_i^{\mathrm{low}} = \frac{\xi_i^{\mathrm{M,low}} h_i}{s_i^{\mathrm{M}}} + \frac{\xi_i^{\mathrm{R,low}} h_i}{s_i^{\mathrm{R}}} + \xi_i^{0,\mathrm{low}}, \tag{7a}$$

where:

$$\xi_i^{\mathrm{M,low}} = N_i^{\mathrm{M}} M_i^{\mathrm{avg}}, \tag{7b}$$
$$\xi_i^{\mathrm{R,low}} = N_i^{\mathrm{R}} \left(S_i^{\mathrm{avg}} + R_i^{\mathrm{avg}}\right), \tag{7c}$$
$$\xi_i^{0,\mathrm{low}} = S_i^{1,\mathrm{avg}} - S_i^{\mathrm{avg}}. \tag{7d}$$

In the same way, the execution time of a job of class i is at most:

$$T_i^{\mathrm{up}} = \frac{\xi_i^{\mathrm{M,up}} h_i}{s_i^{\mathrm{M}}} + \frac{\xi_i^{\mathrm{R,up}} h_i}{s_i^{\mathrm{R}}} + \xi_i^{0,\mathrm{up}}, \tag{8a}$$

where:

$$\xi_i^{\mathrm{M,up}} = N_i^{\mathrm{M}} M_i^{\mathrm{avg}} - 2M_i^{\max}, \tag{8b}$$
$$\xi_i^{\mathrm{R,up}} = N_i^{\mathrm{R}} Sh_{avg}^{typi} - 2Sh_{max}^{typi} + N_i^{\mathrm{R}} R_i^{\mathrm{avg}} - 2R_i^{\max}, \tag{8c}$$
$$\xi_i^{0,\mathrm{up}} = 2S_i^{\max} + S_i^{1,\max} + 2M_i^{\max} + 2R_i^{\max}. \tag{8d}$$

Depending on the guarantees required, it is possible to adopt either a conservative approach to meeting deadlines with T_i^{up} or a less resource-demanding one with T_i^{avg}.

In both cases (upper bounds or approximation), the formulae above reduce to constraints (P1b) and (P2b), by defining $\zeta_i^{0,D} \triangleq \xi_i^0 - D_i < 0$:

$$T_i = \frac{\xi_i^{\mathrm{M}} h_i}{s_i^{\mathrm{M}}} + \frac{\xi_i^{\mathrm{R}} h_i}{s_i^{\mathrm{R}}} + \xi_i^0 \leq D_i \Rightarrow T_i = \frac{\xi_i^{\mathrm{M}} h_i}{s_i^{\mathrm{M}}} + \frac{\xi_i^{\mathrm{R}} h_i}{s_i^{\mathrm{R}}} + \zeta_i^{0,D} \leq 0.$$

B Proofs

Proof of Theorem 1. Since all the constraints of problem (P2) are convex and the Slater constraints qualification holds, the KKT conditions are necessary for optimality. The Lagrangian function of problem (P2) is given by:

$$\begin{aligned}
L =& \delta d + \rho r - \sum_{i\in\mathcal{U}} \frac{p_i}{\Psi_i} + \sum_{i\in\mathcal{U}} \phi_i \left(\frac{\xi_i^{\mathrm{M}}}{s_i^{\mathrm{M}}\Psi_i} + \frac{\xi_i^{\mathrm{R}}}{s_i^{\mathrm{R}}\Psi_i} + \zeta_i^{0,D} \right) \\
&+ \mu_r(r - \bar{r}) + \nu \left[\sum_{i\in\mathcal{U}} \left(\frac{s_i^{\mathrm{M}}}{c_i^{\mathrm{M}}} + \frac{s_i^{\mathrm{R}}}{c_i^{\mathrm{R}}} \right) - r - d \right] \\
&+ \sum_{i\in\mathcal{U}} \left[\mu_i(\Psi_i - \Psi_i^{\mathrm{up}}) + \lambda_i(-\Psi_i + \Psi_i^{\mathrm{low}}) \right] \\
&- \lambda_r r - \lambda_d d - \sum_{i\in\mathcal{U}} \left(\omega_i\, s_i^{\mathrm{M}} + \chi_i s_i^{\mathrm{R}} \right).
\end{aligned}$$

Therefore, the KKT conditions are the following:

$$\delta - \nu - \lambda_d = 0, \tag{9}$$

$$\rho - \nu + \mu_r - \lambda_r = 0, \tag{10}$$

$$\frac{\nu}{c_i^{\mathrm{M}}} - \phi_i \frac{\xi_i^{\mathrm{M}}}{(s_i^{\mathrm{M}})^2\Psi_i} - \omega_i = 0, \quad \forall i \in \mathcal{U}, \tag{11}$$

$$\frac{\nu}{c_i^{\mathrm{R}}} - \phi_i \frac{\xi_i^{\mathrm{R}}}{(s_i^{\mathrm{R}})^2\Psi_i} - \chi_i = 0, \quad \forall i \in \mathcal{U}, \tag{12}$$

$$\frac{p_i}{\Psi_i^2} - \frac{\phi_i}{\Psi_i^2}\left(\frac{\xi_i^{\mathrm{M}}}{s_i^{\mathrm{M}}} + \frac{\xi_i^{\mathrm{R}}}{s_i^{\mathrm{R}}}\right) + \mu_i - \lambda_i = 0, \quad \forall i \in \mathcal{U}, \tag{13}$$

$$\phi_i \left(\frac{\xi_i^{\mathrm{M}}}{s_i^{\mathrm{M}}\Psi_i} + \frac{\xi_i^{\mathrm{R}}}{s_i^{\mathrm{R}}\Psi_i} + \zeta_i^{0,D} \right) = 0, \quad \phi_i \geq 0, \quad \forall i \in \mathcal{U}, \tag{14}$$

$$\mu_r(r - \bar{r}) = 0, \quad \mu_r \geq 0, \tag{15}$$

$$\nu \left[\sum_{i\in\mathcal{U}} \left(\frac{s_i^{\mathrm{M}}}{c_i^{\mathrm{M}}} + \frac{s_i^{\mathrm{R}}}{c_i^{\mathrm{R}}} \right) - r - d \right] = 0, \quad \nu \geq 0, \tag{16}$$

$$\mu_i(\Psi_i - \Psi_i^{\mathrm{up}}) = 0, \quad \mu_i \geq 0, \quad \forall i \in \mathcal{U}, \tag{17}$$

$$\lambda_i(-\Psi_i + \Psi_i^{\mathrm{low}}) = 0, \quad \lambda_i \geq 0, \quad \forall i \in \mathcal{U}, \tag{18}$$

$$\lambda_r\, r = 0, \quad \lambda_r \geq 0, \tag{19}$$

$$\lambda_d\, d = 0, \quad \lambda_d \geq 0, \tag{20}$$

$$\omega_i\, s_i^{\mathrm{M}} = 0, \quad \omega_i \geq 0, \quad \forall i \in \mathcal{U}, \tag{21}$$

$$\chi_i\, s_i^{\mathrm{R}} = 0, \quad \chi_i \geq 0, \quad \forall i \in \mathcal{U}. \tag{22}$$

Constraints (P2b) imply that s_i^{M} and s_i^{R} are positive, hence multipliers ω_i and χ_i are equal to zero. As the adoption of reserved instances is favored, being cheaper than the on demand ones, we obtain $r > 0$ and $\lambda_r = 0$. Furthermore, we have $\nu = \rho + \mu_r \geq \rho > 0$. Therefore, equations (11) guarantee that $\phi_i > 0$ for all $i \in \mathcal{U}$, hence constraints (P2b) hold as equalities.

Finally, we can use equations (P2b), (11) and (12) to compute s_i^{M} and s_i^{R} as a function of Ψ_i. First, we calculate the relation between s_i^{M} and s_i^{R} by using conditions (11) and (12) as follows:

$$\frac{\xi_i^{\mathrm{M}}}{(s_i^{\mathrm{M}})^2\Psi_i} c_i^{\mathrm{M}} = \frac{\xi_i^{\mathrm{R}}}{(s_i^{\mathrm{R}})^2\Psi_i} c_i^{\mathrm{R}} \iff s_i^{\mathrm{M}} = s_i^{\mathrm{R}} \sqrt{\frac{\xi_i^{\mathrm{M}}\, c_i^{\mathrm{M}}}{\xi_i^{\mathrm{R}}\, c_i^{\mathrm{R}}}}.$$

Then, we can replace s_i^{M} by $s_i^{\mathrm{R}}\sqrt{\frac{\xi_i^{\mathrm{M}}\, c_i^{\mathrm{M}}}{\xi_i^{\mathrm{R}}\, c_i^{\mathrm{R}}}}$ into (P2b) to derive an explicit formulation for s_i^{R}, as follows:

$$s_i^{\mathrm{R}} = -\frac{1}{\zeta_i^{0,D}\Psi_i}\left(\sqrt{\frac{\xi_i^{\mathrm{M}}\,\xi_i^{\mathrm{R}} c_i^{\mathrm{R}}}{c_i^{\mathrm{M}}}} + \xi_i^{\mathrm{R}}\right),$$

and along the same lines we can express s_i^{M} in closed-form as follows:

$$s_i^{\mathrm{M}} = -\frac{1}{\zeta_i^{0,D}\Psi_i}\left(\sqrt{\frac{\xi_i^{\mathrm{M}}\,\xi_i^{\mathrm{R}}\,c_i^{\mathrm{M}}}{c_i^{\mathrm{R}}}} + \xi_i^{\mathrm{M}}\right).$$

□

Proof of Theorem 2. Theorem 1 implies that the variables s_i^{M} and s_i^{R} can be written as in equations (3)-(4). Hence, constraint (P3b) is obtained by replacing s_i^{M} and s_i^{R} in constraint (P2d). Moreover, constraints (P2b) can be dropped since they have been used to derive the value of s_i^{M} and s_i^{R}. Hence problem (P3) is equivalent to problem (P2). □

Proof of Theorem 3. First, we notice that the KKT conditions of problem (P3) are necessary and sufficient for optimality since the problem is linear. The KKT conditions can be written as follows:

$$\begin{aligned}
\rho - \nu + \mu_r - \lambda_r &= 0, && (23)\\
\delta - \nu - \lambda_d &= 0, && (24)\\
-p_i + \gamma_i\,\nu + \mu_i - \lambda_i &= 0, \quad \forall\ i\ \in \mathcal{U}, && (25)\\
\nu\left(\sum_{i\in\mathcal{U}} \gamma_i\, h_i^* - r^* - d^*\right) &= 0, && (26)\\
\lambda_r\, r^* &= 0, && (27)\\
\mu_r\,(r^* - \bar{r}) &= 0, && (28)\\
\lambda_d\, d^* &= 0, && (29)\\
\lambda_i\,(h_i^* - H_i^{\mathrm{low}}) &= 0, \quad \forall\ i\ \in \mathcal{U}, && (30)\\
\mu_i\,(h_i^* - H_i^{\mathrm{up}}) &= 0, \quad \forall\ i\ \in \mathcal{U}, && (31)\\
\nu, \lambda_r, \mu_r, \lambda_d &\ge 0, && (32)\\
\lambda_i, \mu_i &\ge 0, \quad \forall\ i\ \in \mathcal{U}. && (33)
\end{aligned}$$

1. Let us assume, by contradiction, that $r^* = 0$. Hence $d^* \ge \sum_{i\in\mathcal{U}} \gamma_i\, h_i^* \ge \sum_{i\in\mathcal{U}} \gamma_i\, H_i^{\mathrm{low}} > 0$, thus $\lambda_d = 0$ and $\nu = \delta$. On the other hand, (28) implies that $\mu_r = 0$ and $\lambda_r = \rho - \nu = \rho - \delta < 0$, which is impossible.
2. Since $r^* > 0$, we have $\lambda_r = 0$, hence (23) implies $\nu = \rho + \mu_r \ge \rho > 0$, thus constraint (P3b) is active at (r^*, d^*, h^*).
3. It follows from (24) that $\nu = \delta - \lambda_d \le \delta$, hence we have $\mu_i = \lambda_i + p_i - \gamma_i\,\nu \ge p_i - \gamma_i\,\nu \ge p_i - \gamma_i\,\delta > 0$. Therefore $h_i^* = H_i^{\mathrm{up}}$.
4. Since $\nu \ge \rho$, we get $\lambda_i = \mu_i + \gamma_i\,\nu - p_i \ge \gamma_i\,\nu - p_i \ge \gamma_i\,\rho - p_i > 0$, hence $h_i^* = H_i^{\mathrm{low}}$.
5. We have $r^* = \sum_{i\in\mathcal{U}} \gamma_i\, h_i^* - d^* \le \sum_{i\in\mathcal{U}} \gamma_i\, H_i^{\mathrm{up}} < \bar{r}$, thus $\mu_r = 0$ and $\nu = \rho$. Therefore, $\lambda_d = \delta - \rho > 0$ implies $d^* = 0$.
6. We have $d^* = \sum_{i\in\mathcal{U}} \gamma_i\, h_i^* - r^* \ge \sum_{i\in\mathcal{U}} \gamma_i\, H_i^{\mathrm{low}} - \bar{r} > 0$, hence $\lambda_d = 0$ and $\nu = \delta$. Therefore, $\mu_r = \delta - \rho > 0$ implies $r^* = \bar{r}$.

□

Proof of the closed-form optimal solution for the two-class case (Table 5). First, we note that $\nu \in [\rho, \delta]$, according to the properties (2) and (3) of Theorem 3. The following implications follow directly from the system (23)–(33):

$$\begin{aligned} \nu < \delta &\implies d^* = 0, \\ \nu > \rho &\implies r^* = \bar{r}, \\ d^* > 0 &\implies r^* = \bar{r}. \end{aligned}$$

The proof is divided into six cases:

1. Let $p_1/\gamma_1 < p_2/\gamma_2 < \rho$. Theorem 3 implies $h_1^* = H_1^{\text{low}}$ and $h_2^* = H_2^{\text{low}}$. If $\bar{r} < \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$, then Theorem 3 guarantees that $r^* = \bar{r}$ and $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - \bar{r}$. If $\bar{r} > \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$, then
$$r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - d^* \leq \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} < \bar{r},$$
thus $d^* = 0$ and $r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$.
2. Let $p_1/\gamma_1 < \rho < p_2/\gamma_2 < \delta$. Theorem 3 implies $h_1^* = H_1^{\text{low}}$. We distinguish three cases.
   a) If $\nu \in [\rho, p_2/\gamma_2)$, then $\mu_2 \geq p_2 - \gamma_2 \nu > 0$ hence $d^* = 0$, $h_2^* = H_2^{\text{up}}$ and $r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} \leq \bar{r}$.
   b) If $\nu = p_2/\gamma_2$, then $d^* = 0$, $r^* = \bar{r}$ and $h_2^* = \left(\bar{r} - \gamma_1 H_1^{\text{low}}\right)/\gamma_2$. In particular, we have $\gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} \leq \bar{r} \leq \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$.
   c) If $\nu \in (p_2/\gamma_2, \delta]$, then $\lambda_2 \geq \gamma_2 \nu - p_2 > 0$ thus $r^* = \bar{r}$, $h_2^* = H_2^{\text{low}}$ and $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - \bar{r}$. In particular, we have $\bar{r} \leq \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$.
3. Let $p_1/\gamma_1 < \rho < \delta < p_2/\gamma_2$. It is similar to case 1. We have $h_1^* = H_1^{\text{low}}$ and $h_2^* = H_2^{\text{up}}$. If $\bar{r} < \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$, then $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} - r^* > 0$, thus $r^* = \bar{r}$ and $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} - \bar{r}$. If $\bar{r} > \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$, then $r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} - d^* < \bar{r}$, thus $d^* = 0$ and $r^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$.
4. Let $\rho < p_1/\gamma_1 < p_2/\gamma_2 < \delta$. We distinguish five cases.
   a) If $\nu \in [\rho, p_1/\gamma_1)$, then $d^* = 0$. Furthermore, $\mu_1 \geq p_1 - \gamma_1 \nu > 0$ and $\mu_2 \geq p_2 - \gamma_2 \nu > 0$, hence $h_1^* = H_1^{\text{up}}$ and $h_2^* = H_2^{\text{up}}$. Thus, $r^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}} \leq \bar{r}$.
   b) If $\nu = p_1/\gamma_1$, then $d^* = 0$ and $r^* = \bar{r}$. Since $\mu_2 \geq p_2 - \gamma_2 \nu > 0$, we get $h_2^* = H_2^{\text{up}}$ and $h_1^* = [\bar{r} - \gamma_2 H_2^{\text{up}}]/\gamma_1$. In particular, we have $\gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} \leq \bar{r} \leq \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$.
   c) If $\nu \in (p_1/\gamma_1, p_2/\gamma_2)$, then $d^* = 0$ and $r^* = \bar{r}$. Since $\lambda_1 > 0$ and $\mu_2 > 0$, we have $h_1^* = H_1^{\text{low}}$ and $h_2^* = H_2^{\text{up}}$. Thus, $\bar{r} = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$.
   d) If $\nu = p_2/\gamma_2$, then $d^* = 0$ and $r^* = \bar{r}$. Since $\lambda_1 > 0$, we get $h_1^* = H_1^{\text{low}}$ and $h_2^* = \left(\bar{r} - \gamma_1 H_1^{\text{low}}\right)/\gamma_2$. In particular, we have $\gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} \leq \bar{r} \leq \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$.
   e) If $\nu \in (p_2/\gamma_2, \delta]$, then $r^* = \bar{r}$. Furthermore, $\lambda_1 > 0$ and $\lambda_2 > 0$ thus $h_1^* = H_1^{\text{low}}$, $h_2^* = H_2^{\text{low}}$ and $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}} - \bar{r}$. In particular, we have $\bar{r} \leq \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{low}}$.
5. Let $\rho < p_1/\gamma_1 < \delta < p_2/\gamma_2$. It is similar to case 2. We have $h_2^* = H_2^{\text{up}}$ and distinguish three cases:
   a) If $\nu \in [\rho, p_1/\gamma_1)$, then $d^* = 0$, $h_1^* = H_1^{\text{up}}$ and $r^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}} \leq \bar{r}$.
   b) If $\nu = p_1/\gamma_1$, then $d^* = 0$, $r^* = \bar{r}$ and $h_1^* = \left(\bar{r} - \gamma_2 H_2^{\text{up}}\right)/\gamma_1$. In particular, we have $\gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} \leq \bar{r} \leq \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$.
   c) If $\nu \in (p_1/\gamma_1, \delta]$, then $r^* = \bar{r}$, $h_1^* = H_1^{\text{low}}$ and $d^* = \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}} - \bar{r}$. In particular, we have $\bar{r} \leq \gamma_1 H_1^{\text{low}} + \gamma_2 H_2^{\text{up}}$.
6. Let $\delta < p_1/\gamma_1 < p_2/\gamma_2$. It is similar to case 1. We have $h_1^* = H_1^{\text{up}}$ and $h_2^* = H_2^{\text{up}}$. If $\bar{r} < \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$, then $d^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}} - r^* > 0$, thus $r^* = \bar{r}$ and $d^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}} - \bar{r}$. If $\bar{r} > \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$, then $r^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}} - d^* < \bar{r}$, thus $d^* = 0$ and $r^* = \gamma_1 H_1^{\text{up}} + \gamma_2 H_2^{\text{up}}$.

$\square$