Computers and Operations Research 81 (2017) 231–246

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Optimal distributed task scheduling in volunteer clouds

Stefano Sebastio [a,*], Giorgio Gnecco [b], Alberto Bemporad [b]

[a] LIMS London Institute of Mathematical Sciences, W1K 2XF London, UK
[b] IMT Institute for Advanced Studies, 55100 Lucca, Italy

ARTICLE INFO

Article history:
Received 14 September 2015
Revised 16 June 2016
Accepted 5 November 2016
Available online 14 November 2016

Keywords:
Cloud computing
Distributed optimization
Integer programming
Combinatorial optimization
ADMM

ABSTRACT

The ever increasing request of computational resources has shifted the computing paradigm towards solutions where less computation is performed locally. The most widely adopted approach nowadays is represented by cloud computing. With the cloud, users can transparently access to virtually infinite resources with the same aptitude of using any other utility. Next to the cloud, the volunteer computing paradigm has gained attention in the last decade, where the spared resources on each personal machine are shared thanks to the users' willingness to cooperate. Cloud and volunteer paradigms have been recently seen as companion technologies to better exploit the use of local resources. Conversely, this scenario places complex challenges in managing such a large-scale environment, as the resources available on each node and the presence of the nodes online are not known a-priori. The complexity further increases in presence of tasks that have an associated Service Level Agreement specified, e.g., through a deadline. Distributed management solutions have then be advocated as the only approaches that are realistically applicable.

In this paper, we propose a framework to allocate tasks according to different policies, defined by suitable optimization problems. Then, we provide a distributed optimization approach relying on the Alternating Direction Method of Multipliers (ADMM) for one of these policies, and we compare it with a centralized approach. Results show that, when a centralized approach can not be adopted in a real environment, it could be possible to rely on the good suboptimal solutions found by the ADMM.

1. Introduction

The ever growing demand of computational resources has shifted users from local computation on personal devices towards the adoption of centralized *cloud computing* technologies. Using the virtualized resources available on remote data centers, the cloud allows the access to virtually unlimited computational resources with the same aptitude of using any other utility service.

Together with the latest advance on virtualization technologies adopted by the cloud, in the last decade also the CPU manufacturers brought a great leap forward in performance starting the *multi-core era* even for the personal devices. Nowadays, desktop and laptop devices have resources largely unused for great part of the time (e.g., during web-browsing or text-editing activities) while having possibly scarce resources for other activities (e.g., for large-scale simulations or graphic-editing). These scenarios have opened the door to the growth of another ICT trend of the last years: the *volunteer computing* paradigm. Such a paradigm foresees the use of the spared resources on personal devices by all other network participants, relying on the users' willingness to cooperate. Each user shares a quote of its unused resources with other users, and receives other shared resources when he needs more than the local resources at his disposal. Therefore, the volunteer network turns out to be a heterogeneous (in terms of shared resources) and highly dynamic (not knowing a-priori the presence online of the volunteers) large-scale system (usually constituted from hundreds to thousands of nodes).

The cloud can then be enhanced through the combination with the volunteer paradigm. The combination is referred as the *volunteer cloud* [1]. Indeed, in specific application domains, such as latency dependent environments, the cloud still suffers its centralized nature where, obviously, its data-centers can not be located close to all the users. Examples of such domains have been described by Microsoft [2] as *cloudlets*, where the computational burden on mobile devices running virtual- or augmented-reality applications can be alleviated relying on low-latency (i.e., closely located) more powerful devices. The volunteer cloud should be considered [3] as a companion force to possibly enhance the traditional cloud in specific domains such as computing- and storage-oriented applications (e.g., Map-Reduce and streaming applications have been deployed using this paradigm in [4]). On the other hand, the volatile presence of the nodes makes it not suitable for service-oriented applications. The growing interest and success of

* Corresponding author.
E-mail address: stefano.sebastio@alumni.imtlucca.it (S. Sebastio).

http://dx.doi.org/10.1016/j.cor.2016.11.004
0305-0548/

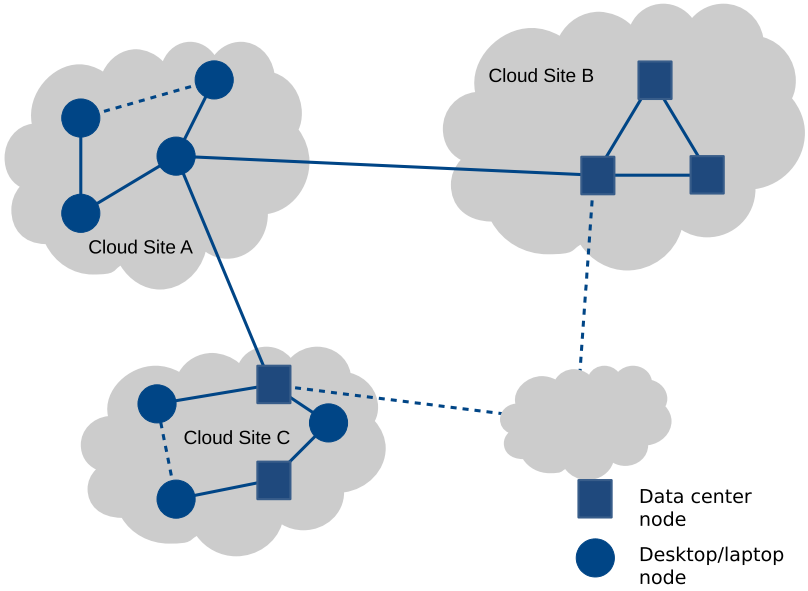


Fig. 1. Example of a volunteer cloud network: each cloud site can be constituted by data centers, personal devices or both (from [10]).

the volunteer cloud is witnessed by the large number of existing platforms and projects based on such a paradigm, e.g., BOINC [5], HTCondor [6], OurGrid [7], Seattle [8] and SETI@home [9].

Typically, the volunteer nodes are organized and can communicate with each other through a Peer-to-Peer (P2P) overlay network. An example of the volunteer cloud network is depicted in Fig. 1. Despite centralized managing solutions are currently widely adopted, the inherent characteristics of the volunteer cloud network makes their distribution of execution-oriented applications inefficient. Indeed, in such scenario a central node can constitute a bottleneck without being able to have updated knowledge on the nodes characteristics and their actual load in order to perform a global optimization while scheduling tasks with heterogeneous resource and performance requests. Distributed managing solutions have then been advocated as the only practical solution. Several distributed computing approaches [11–13] based on autonomic (i.e., self-managing) agents [14] have been explored in the literature. We only mention some works without dwelling on them since all the proposed solutions are *pure heuristic* solutions e.g., relying on Ant Colony Optimization [15,16], Evolutionary Algorithm [17], Spatial computing [18] or Reputation-based [19] techniques.

The present work aims at proposing a framework to distribute task execution requests, according to different policies, each formalized as a mathematical optimization problem solved in a distributed fashion. In particular, we focus on the Alternating Direction Method of Multipliers (ADMM) [20] to decompose the optimization problem, which is then distributed to and independently solved by the volunteer nodes.

Synopsis. The paper is structured as follows. Section 2 presents related work and particularly Section 2.1 briefly presents the ADMM. Our volunteer cloud model is presented in Section 3 while the optimization problem is formalized in Section 4. One of the policies allowed by our framework is accurately illustrated in Section 5 and solved with a centralized solution and with two different distributed ADMM variants. The considered scenario, its *Matlab* implementation relying on *CVX* [21] and Gurobi [22], is discussed in Section 6 together with results and a numerical comparison of the proposed solutions. The description of the objective functions for the other policies of our framework is presented in Section 7. Finally, Section 8 concludes the work with some final remarks and outlines other current and future research efforts.

2. Related work

Recently, the volunteer cloud has been modeled in [23] (called the cloud-assisted approach therein). Criticality and infeasibility of a centralized solution, to determine if a subset of tasks would be better executed by the cloud or by the volunteers, are highlighted in that paper referring to the problem as a *cloudy knapsack problem* i.e., an online knapsack problem but in presence of a limited knowledge on the current state (*cloudy view*). Despite in the paper motivation the authors argue for the need of a distributed decision making, later in the work the focus is only on the characterization of the knapsack problem where the available budget (corresponding to the available resources) is known only in terms of a probability distribution. Authors prove an upper bound on the performance of their knapsack algorithm where a probability distribution models the budget uncertainty.

In [24] authors accurately model the hybrid cloud environment (i.e., with some infrastructures operating solely for an organization while others provided in a multi-tenant environment) as a mixed-integer nonlinear programming problem specified in AMPL [25]. The authors' goal is the minimization of the monetary cost to run a set of both compute- and data-intensive tasks. Actually, their implementation takes even the deadline into account in the cost function, although not in a explicit way. Indeed, their cost function mixes the time for completing the tasks with the actual price for using a given service. Although there are some similarities with our model, the major differences are represented by their indifference to distributed solutions and by not considering the actual load on machines.

The above mentioned model is further extended in [26] to deal with workflows instead of independent tasks. The authors define the workflow as a directed acyclic graph and then group the tasks into levels. In each level, tasks are independent among them and intermediate deadlines are assigned.

DONAR [27] is a distributed system that aims at mapping replica selection in the cloud, i.e., the choice of a particular location based on performance, load and cost in presence of geo-replicated services. Its optimization problem is designed in a way to be easily decomposed into independent subproblems that can

be solved locally (since even the constraints can be split on the nodes). The global constraint (related to the bandwidth cap) could be temporarily violated by the intermediate solutions of the algorithm. Moreover, the problem turns out to be a convex quadratic programming problem (with the objective function continuously differentiable and convex on the entire set of variables) solved with the dual decomposition method, where the optimization variables are constituted by the portions of traffic load that should be mapped from each client to each node. A similar problem is dealt in [28]. There a parallel implementation of ADMM is adopted to solve a convex optimization problem that has been validated on a trace-driven simulation relying on the Wikipedia requests trace.

DONAR is extended in [29] considering energy efficient solutions as a "first-class" design constraint (i.e., a priority) alongside performance, while distributing tasks in the cloud. In particular, the objective function takes into account the energy costs in the presence of data-intensive services deployed in the cloud. In more details, the objective function of the convex optimization problem in [29] is constituted by the sum of local objective functions, and the problem is solved using the dual decomposition and the consensus-based distributed projected subgradient method (CDPSM) [30].

The task allocation problem in presence of hard deadlines and classes of priorities has been studied in [31] for powertrain automotive applications. Their problem formulation turns out to be a mixed integer linear programming problem solved using CPLEX [32]. The different characteristics of their domain of application makes difficult a direct comparison with our approach, despite it is possible to recognize that their solution assumes a centralized approach where an updated global knowledge on the load of the nodes is available (since their problem does not need to deal with a large-scale system).

Main contributions. All in all, our paper contributes to the existing literature related to the volunteer cloud in that it proposes:

- a framework to accommodate different task distribution policies in large-scale distributed cloud environments;
- a mathematical formulation of the optimization problem associated with such a framework, which is driven by real system requirements of the volunteer cloud and takes into account various issues, such as, FIFO (First In, First Out) queue, tasks with deadlines, the actual load on the machines, the need to adopt a distributed solution;
- the application of the distributed ADMM for solving the above optimization problem.

2.1. The ADMM: alternating direction method of multipliers

The increasing size and complexity of many datasets pose challenges to analyze such problems especially when the data under analysis originate from decentralized sources. Distributed solutions are thus advocated as promising and desirable to work in such large-scale domains.

Recently the *Alternating Direction Method of Multipliers (ADMM)* is gaining momentum [20] in the field of distributed optimization, despite the method has been developed almost fifty years ago. The ADMM is an algorithm to solve *convex optimization* [33] problems in a distributed/parallelized fashion adopting a *decomposition-coordination* method, in which the problem is subdivided into small problems that can be solved independently, under the "coordination" of a central node, to solve the original problem. The method has its roots on the methods of dual decomposition and augmented Lagrangian.

The method is usually applied to convex optimization problems of the form:

$$\text{minimize } f(x) + g(z), \tag{1a}$$

$$\text{subject to } Ax + Bz = c, \tag{1b}$$

where $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ and $g : \mathbb{R}^m \to \overline{\mathbb{R}}$ are convex functions, and $x \in \mathbb{R}^n, z \in \mathbb{R}^m, A \in \mathbb{R}^{p\times n}, B \in \mathbb{R}^{p\times m}, c \in \mathbb{R}^p$. Here, $\overline{\mathbb{R}}$ denotes the extended real number line.

Then, given the *penalty parameter* $\rho > 0$ and $y \in \mathbb{R}^p$, the augmented Lagrangian is expressed as:

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2, \tag{2}$$

and, finally, each iteration of the ADMM is defined as follows:

$$x^{k+1} := \underset{x}{\operatorname{argmin}}\, L_\rho(x, z^k, y^k), \tag{3a}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}}\, L_\rho(x^{k+1}, z, y^k), \tag{3b}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c), \tag{3c}$$

where the penalty parameter in the augmented Lagrangian constitutes the *step size* while updating the *dual variable* y. ADMM is often transformed in the equivalent *scaled* form for convenience of implementation, resulting in a more compact expression. The scaled form of ADMM is expressed scaling the dual variable as $u = \frac{1}{\rho}y$.

We refer the interested reader to [20] for the proof of convergence (under mild conditions) and methods to compute the stopping criterion of the ADMM. A method for the optimal choice of the ρ parameter in ADMM that ensure the smallest possible convergence factor for quadratic problems is provided in [34].

If the problem is not convex, it is still possible to apply ADMM, despite this should be just considered as a heuristic in that case [20]. An extension of the ADMM for nonconvex problems through the use of heuristics is proposed in [35]. There, integer and continuous variables of the problem are identified, and the source of non convexity are the first variables. The continuous relaxation of the problem is initially considered obtaining bounds on the optimal solution value of the original problem and thus using these hints to reduce the search space. The basic idea is to start with ADMM and integer variables, but once these variables are observed not to change values for some iterations, their value is fixed (so actually excluding them from the variables set and changing them to constants), to simplify the following ADMM iterations that could refine the solution for the continuous variables. Their *Release and Fix* approach swings from the original to the simplified problem (i.e., without the integer variables) at each iteration. Numerical results demonstrate that their distributed approach can reach a 1% relative optimality gap with a reduced solution time compared to a centralized solution, while it is executed on a computer cluster.

Another interesting application of ADMM for nonconvex problems in computer vision is studied in [36]. There, the authors adopt a tree-based decomposition of discrete labeling problems in large-scale random fields. The decomposition exploits the submodularity of the non-convex subproblem functions allowing to solve them exactly. Despite the lack of guarantees on convergence for the original problem, their empirical results are encouraging. A parallel implementation relying on both CPU and GPU cores allowed the authors of [36] to obtain a significant speed-up while benchmarking low-level vision problems such as stereo correspondence, image segmentation and denoising.

3. The volunteer cloud computing model

The volunteer cloud environment is constituted by a large number of nodes that share their computational and/or storage resources with the other network participants. Usually the cloud architecture relies on a centralized load balancer which collects information from and distributes commands to the nodes. Unfortunately, in the volunteer cloud, the large-scale nature of such a kind of network (in the order of hundreds or thousands of nodes) and the volatility of the nodes participation (i.e., it is not possible to know a-priori the time a node will join or leave the network), makes the dissemination of resources and load characteristics to a central load balancer a hard, if not impossible, job. Consequently, tasks execution requests (whose characteristics could be diversified in terms of both required resources and service performance) cannot be distributed by solving an optimization problem in a completely centralized way.

Distributed optimization techniques are then advocated as promising alternatives. In this work we focus on such a technique, to optimize the task allocation among the volunteer nodes. This section presents the volunteer cloud computing model that will be used in Section 4 to define the optimization problem of interest. Since our focus is on the distribution of tasks execution requests, the main component of our model are the tasks (Section 3.1) and the resources (Section 3.2). In such a large-scale decentralized environment each node acts as both producer and consumer of tasks execution requests.

3.1. Tasks

Application programs are modeled as sets of independent *tasks*. Each task is defined by its duration, the degree of parallelism that it is able to exploit, and its minimum memory requirement. Our model assumes that the task duration can be predicted with accuracy. Moreover, we assume a perfect linear speed-up for the parallelism.

Definition 3.1 (task)**.** A *task* is a tuple $\langle \delta, \rho, \mu \rangle$, where $\delta \in \mathbb{N}^+$ is the task duration (expressed in clock cycles), $\rho \in \mathbb{N}^+$ is the degree of parallelism of the task, and $\mu \in \mathbb{N}^+$ is the memory requirement of the task.

Adopting a perfect linear speed-up model, a degree of parallelism ρ means that a task with duration δ can be ideally executed with a CPU that has ρ computational units (e.g., cores) in $\lceil \frac{\delta}{\rho} \rceil$ consecutive clock cycles. If less than ρ computational units are available, the time to execute the task will be bounded in terms of the number of available units (as expressed in the following Eq. (4)). For typical tasks, the relative error between $\frac{\delta}{\rho}$ and $\lceil \frac{\delta}{\rho} \rceil$ is close to 0. Therefore, in the following, for simplicity of notation, $\lceil \frac{\delta}{\rho} \rceil$ is replaced by $\frac{\delta}{\rho}$.

Definition 3.2 (task execution request)**.** A *task execution request* is a tuple $\langle \delta, \rho, \mu, \tau_a, \tau_d \rangle$, where $\langle \delta, \rho, \mu \rangle$ is a task, $\tau_a \in \mathbb{R}^+$ is the task arrival date, and $\tau_d \in \mathbb{R}^+$ is its termination deadline. The deadline can constitute one of the parameters specified in the task Service Level Agreement (SLA) [37].

We assume that for each task a single execution is required, i.e., it is enough that a single node takes the execution request in charge.

3.2. Virtual resources

The volunteer nodes provide an isolated environment for each task execution request through a Virtual Machine (VM), modeled by memory and processing units. In our scenario, the main feature of interest of the VMs is the latter since our main concern is the running time of tasks, which we consider to be computation-intensive (rather than data- or communication-intensive). We assume that, given the voluntary participation of the nodes and the not dedicate use of the physical machine (only spare resources are shared by users), each node runs a single VM. Therefore, to provide an isolated environment for each user, tasks execution requests are processed one at a time. For the sake of simplicity we consider that VMs have homogeneous processing units, i.e., all the cores in the same VM have the same clock frequency.

Definition 3.3 (Virtual Machine, VM)**.** A *VM* is a tuple $\langle \kappa, \phi, \nu \rangle$ where $\kappa \in \mathbb{N}^+$ is the number of cores, $\phi \in \mathbb{N}^+$ is their frequency, and $\nu \in \mathbb{N}^+$ is the amount of memory.

Definition 3.4 (Execution time)**.** The execution time $E(T, VM)$ required for completing a task $T = \langle \delta, \rho, \mu \rangle$ on a virtual machine $VM = \langle \kappa, \phi, \nu \rangle$ whose cores have frequency ϕ is defined by the following equation:

$$E(T, VM) = E(\delta, \phi, \rho, \kappa) = \frac{\delta}{\phi \cdot \min\{\rho, \kappa\}}, \tag{4}$$

i.e., the task duration on a single core which is equal to δ/ϕ is divided by the maximum degree of parallelism that can be exploited, which is bounded by both the amount of available cores (κ) and the degree of parallelism of the task (ρ).

A task $T = \langle \delta, \rho, \mu \rangle$ can be executed on a virtual machine $VM = \langle \kappa, \phi, \nu \rangle$ only if the following memory requirement constraint is satisfied:

$$\mu \leq \nu. \tag{5}$$

Tasks accepted for execution are added to the node execution queue and executed according to a FIFO policy. If not otherwise specified, a machine *VM* can accept a task *T* in its execution queue only if it can respect the task deadline.

4. The optimization problem

In our model, resource capacities (i.e., memory and cores characteristics) and tasks already on the node execution queue pose limits on the following task execution requests that a node can accept respecting the associated SLAs.

In a first approximation, task execution requests can be dispatched periodically according to an *allocation period*. In each period, task requests should be distributed optimizing with respect to an allocation policy defined by the volunteer cloud manager. All the tasks generated during an allocation period are assigned to the subsequent *task set* and dispatched in the next allocation period. The duration of the period itself constitutes a parameter which can vary the performance of the system. A short period could allow a more timely allocation of the tasks while resulting more computationally expensive. Conversely, a large period can possibly reduce the chance to satisfy task requests that have more stringent deadlines, despite they could be less computational demanding.

This section presents the data structures (Section 4.1) used in the definition and in the resolution of our optimization problem. Moreover a brief description of some allocation policies that can be implemented within our framework is presented at the end of the section (Section 4.2).

4.1. Data structures

In each allocation period *t*, having *K* tasks and *N* nodes, the data structures used hereinafter (following the notations used in the definitions in Sections 3.1 and 3.2) are as follows:

- $taskSet = \begin{pmatrix} \tau_{a1} & \tau_{d1} & \rho_1 & \delta_1 & \mu_1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \end{pmatrix} \in \mathbb{R}^{K\times 5}$, where the tasks are sorted according to their deadline τ_d, from closest to farthest (this sorting should increase the chance to execute more tasks respecting their deadline, see the notes at the end of this section).
- $nodeSet = \begin{pmatrix} \phi_1 & \kappa_1 & \nu_1 \\ \vdots & \cdots & \vdots \end{pmatrix} \in \mathbb{R}^{N\times 3}$, with the characteristics of the VMs.
- $freeTimes = \begin{pmatrix} free_1 \\ \vdots \end{pmatrix} \in \mathbb{R}^{N}$, with the times at which each node completes the execution of all the tasks that are already in its execution queue (i.e., execution requests that have been assigned to the nodes in the previous allocation periods but that have not yet been completed).
- $X(:= taskAlloc) \in \mathbb{Z}_2^{K\times N}$ is the *task allocation matrix*, a binary matrix ($\mathbb{Z}_2 = \{0, 1\}$) where the element x_{ij} is equal to 1 if the task i is executed on node j and 0 otherwise. Thus, reading it by column one gets the tasks accepted by each node in the current allocation period; while, reading it by row one gets the node(s) that has (have) accepted the task for execution, if any. This matrix represents the *decision variable* of our optimization problem.
- $execEnd_X \in \mathbb{R}^K$, collects the times at which each task execution is completed, taking into account the subset of tasks each node has accepted for execution in the current allocation X (a more formal definition and a step-to-step computation is given in the following, see Section 5). The components of $execEnd_X$ assume value 0 for the tasks that are not accepted by any node.
- $executedTasks \in \mathbb{N}^N$, collects the number of tasks, belonging to the current allocation period, that each node has taken in charge to execute. Each component $executedTasks_j$ is expressed as $\sum_{l=0}^{K-1} X(l, j)$.

In each allocation period, tasks are sorted and evaluated for execution according to their deadline, thus the considered allocation scheduling policy is the *EDF* (Earliest Deadline First). While, as stated before, once the tasks have been accepted, they are executed according to a FIFO policy. This choice is dictated by the willingness to increase the chance to accept as many tasks as possible (respecting their deadlines), while maintaining a low complexity, avoiding to rearrange the execution queue every time new tasks are accepted for execution in the next allocation period.

It is worth noting that the choice of the scheduling policies (for both allocation and execution) affects the selection of tasks that are assigned to the nodes. Indeed, a different sorting in the tasks to be executed or allocated could bring to a different time for completing the tasks ($execEnd_X$) and in turn to a different allocation of the tasks (X). In other terms, with a different sorting of the execution requests a *feasible* allocation could become *unfeasible* and vice-versa.

For instance, let us consider a long-running task with a relaxed deadline that is evaluated for allocation before other small-running tasks but with high demand on deadline: the first task can constitute a bottleneck for all other tasks, which could not be executed respecting their SLAs. A simple tasks sorting (where the tasks with a high demand on deadline are evaluated first) could allow to execute more tasks, while the tasks with a relaxed deadline could be executed later without any effect on their SLA. Obviously, other, more sophisticated and effective scheduling disciplines could also be implemented and evaluated.

4.2. Allocation policies

The objective function considered in our framework could take into account different performance metrics while allocating the tasks to the nodes. In this section we briefly describe five different allocation policies:

- *Number of executed task*: it attempts to maximize the number of executed tasks (Section 5).
- *Termination time*: it attempts to minimize the time at which the execution ends for the entire task set (Section 7.1).
- *Response time*: it attempts to minimize the sum of the response times for each task in the set, even ignoring the SLA if needed. Thus, all the execution requests are assigned (Section 7.2).
- *Fair strategy*: it attempts to evenly distribute the tasks among the participant nodes (Section 7.3).
- *Green strategy*: it attempts to consolidate the tasks among few participant nodes (Section 7.4).

Each of the above mentioned policies, during the formalization of the optimization problem, will require one or a subset of the *constraints* which are listed here:

(a) each node can execute a task only if it has enough memory:

$$\mu_l \le \nu_j \quad \forall\ task\ l\ executed\ on\ a\ node\ j\ ; \tag{6}$$

(b) each task must be assigned to one and only one node, even if there is no node that can satisfy its deadline requirement:

$$\sum_{j=0}^{N-1} X(l, j) \ = \ 1 \quad \forall\ l \ \in \ \{0, 1, \ldots, K-1\}. \tag{7}$$

If it is impossible to satisfy all the tasks deadlines, other node policies will discern which tasks should be executed first and which ones discarded or completed lately (i.e., on a *best-effort* basis not fulfilling the deadline specified in the SLA);

(c) each task, if executed, should complete respecting its deadline:

$$execEnd_X(l) \le \tau_d(l) \quad \forall\ task\ l, \tag{8}$$

where τ_d is a column vector with the task deadlines;

(d) each task can be executed by only one node:

$$\sum_{j=0}^{N-1} X(l, j) \ \le 1 \quad \forall\ l \ \in \ \{0, 1, \ldots, K-1\}. \tag{9}$$

It is worth noting that some of the above mentioned constraints, on certain conditions, could prove to be mutually exlusive (i.e., (b) and (c)) while constraint (d) relaxes constraint (c). Obviously, in these situations, those constraints are not placed at the same time.

5. Maximization of the number of executed tasks

In this section we focus on the accurate description of one of the policies of our framework, namely, the maximization of the number of executed tasks. Without loss of generality other policies can be implemented straightforwardly as exemplified later in Section 7.

The policy described in this section aims at optimizing the number of tasks that are executed respecting their deadline. The associated optimization problem can then be written as the maximization of:

$$\sum_{j=0}^{N-1} \sum_{l=0}^{K-1} taskAlloc(l, j), \tag{10}$$

where the inner summation counts the tasks executed on the node j. The constraints (a), (c) and (d) must be met.

Thus, more formally, the optimization problem is:

$$\text{maximize} \sum_{j=0}^{N-1} \sum_{l=0}^{K-1} X(l,j), \tag{11}$$

subject to:

$$\sum_{j=0}^{N-1} (\text{diag}(\mu) \cdot X)(l,j) \leq X \cdot \nu \qquad \forall\ l \in \{0, \ldots K-1\}, \tag{12a}$$

$$\sum_{j=0}^{N-1} X(l,j) \leq 1_K \qquad \forall\ l \in \{0, \ldots K-1\}, \tag{12b}$$

$$execEnd_X(l) \leq \tau_d(l) \qquad \forall\ task\ l \in \{0, \ldots K-1\}, \tag{12c}$$

where all the inequalities are component-wise comparisons and the $\cdot$ symbol denotes the matrix product, while $\text{diag}(\mu)$ is a diagonal square matrix in which the elements on the main diagonal are the memory requirements for the tasks in the current allocation period. The left side of Eq. (12c), with the implicit dependency from X, expresses the time required to complete the execution of each task l according to the node j that has taken in charge its execution, if any.

Throughout the paper, if not otherwise specified, the index l refers to a task, j refers to a volunteer node, while i refers to one of the constraints related to the optimization problem under study.

The above mentioned optimization problem has been solved in *Matlab* relying on the *CVX* modeling system, which is based on the Disciplined Convex Programming ruleset (DCP). The DCP ruleset [38] allows one to describe problems that are convex by construction. Objective functions and constraints are expressed relying on a set of basic atoms and a restricted set of operation rules that preserve convexity. In this way, DCP is able to keep track of affine, convex and concave expressions. If a problem does not adhere to the DCP it is rejected even if it is convex, the converse instead can not happen (i.e., a problem recognized as convex by DCP is always convex). *CVX* [21,38] is a modeling system that, relying on DCP, is able to simplify the task of specifying the problem for: linear (LPs), quadratic (QPs), second-order cone (SOCPs) and semidefinite (SDPs) programs. Recently, *CVX* has been extended to support *mixed-integer* models where one or more variables have integer or binary values (as X in our problem). These models are defined as *Mixed-Integer Disciplined Convex Problems (MIDCPs)*. Strictly speaking, these are nonconvex optimization problems for which, if the integrality constraint is removed, one obtains a Disciplined Convex Problem.

While all other constraints can be written straightforwardly, the implementation of Eq. (12c) in *CVX* requires some computations to respect the DCP, since the ruleset implies that convexity should be an explicit goal of the modeling process itself. In the following we describe the computations required to express the last constraint in *CVX*. Recalling the definition of execution time in Eq. (4), we construct a matrix e whose generic element e_{lj} represents the computational time required to execute the task l on node j.

A row vector *initTime* with the times at which each node can start the execution in the current allocation period, can be built considering the current time t and the residual computation from the previous time periods (*freeTimes*):

$$initTime := \max\{t, freeTimes\} \in \mathbb{R}^{1 \times N}. \tag{13}$$

Thus, in order to evaluate the time required to execute the tasks in the current period, the node should consider the computation in progress and the tasks already in its queue (and accepted for execution in the previous allocation periods). An intermediate step of computation requires adding *initTime* to the tasks execution times:

$$timeToExecTask = e + \begin{pmatrix} initTime \\ 0 \end{pmatrix} = \begin{pmatrix} e_{11} + it_1 & e_{12} + it_2 & \cdots \\ e_{21} & e_{22} & \cdots \\ \vdots & \ddots & \vdots \end{pmatrix}, \tag{14}$$

where $0_{K-1,N}$ is a zero matrix, while it_j are the elements of the vector *initTime*.

Then, assuming a FIFO policy during the task execution, if every task was executed by each node, one would obtain:

$$execCum = cumulativeSum(timeToExecTask) = \begin{pmatrix} e_{11} + it_1 & e_{12} + it_2 & \cdots \\ e_{21} + e_{11} + it_1 & e_{22} + e_{12} + it_2 & \cdots \\ \vdots & \ddots & \vdots \end{pmatrix}, \tag{15}$$

where the *cumulativeSum* function (*cumsum* in *Matlab*) returns a matrix containing the cumulative sums for each column starting from the first row.

Defining $\overline{X}$ as the *complement matrix* of the binary matrix X with the tasks that are not executed by the nodes, with further computation it is possible to compute the time at which the execution of each task is completed ($execEnd_X$) according to the node that has taken it in charge, if any:

$$execRemovedNotExec = execCum - \overline{X}. * execCum, \tag{16a}$$

$$finishingTimeDirty = execRemovedNotExec - cumulativeSum(\overline{X}. * e), \tag{16b}$$

$$execEnd_X = finishTime = \max_{\forall\ row}\{0, \max_{\forall\ row}\{finishingTimeDirty\}\} = \sum_{by\ row} (finishingTimeDirty^+), \tag{16c}$$

where $.*$ is the element-by-element product of the matrices. Eq. (16a) sets to zero the cells for the tasks that are not executed by the corresponding node in Eq. (15). However, this operation is not enough to obtain the time needed to complete the tasks executions. Indeed, observing Eq. (15), it is possible noting that since we are considering a FIFO policy, setting to zero the cells for the tasks not executed is not enough. For instance, if the first node executes the second but not the first task, with Eq. (16a) the first cell is set to zero but the element in the second row first column still maintains a dependency from a task that is not actually executed (e_{11} in this example). Eq. (16b) is introduced to remove these undesired dependencies. Conversely, this operation builds a *dirty* matrix, since its matrix elements could have negative values while subtracting from cells that where already set to zero from the first operation (Eq. (16a)). Finally, Eq. (16c) builds the tasks finishing times for the tasks that are actually executed, considering the nodes that have accepted their executions (where the $(\cdot)^+$ operator chops off the negative values). Note that the last equality in Eq. (16c) is obtained considering that for each row there can be at most one positive element.

Other simpler operations could have led to the same result but requiring, at some point, the product between elements of the (matrix) optimization variable X. For instance, a possible easily attained computation of $execEnd_X$ could be given by $\sum_{by\ row}(cumulativeSum(e. * X). * X) + X * initTime^T$, where the *cumulativeSum* is needed by the FIFO queue and the second

element-by-element product sets to zero the matrix elements corresponding to the tasks not executed by a node. Unfortunately, despite in our definition X is binary, this operation brings the expression to be not convex as well as not comply with the DCP ruleset.

Note on the Optimization Problem

The problem formulated above (Eqs. (10)–(12c)) can be seen as an *Integer Programming (IP)* problem (since the elements of X assume integer values) whose goal consists in finding an optimal allocation of the tasks. With some implementation effort, particularly when K and/or N are large, Eq. (16c) can be reformulated substituting the *max* operator with linear inequalities, therefore obtaining an Integer Linear Programming (ILP) problem. ILP problems are well known to be *NP-complete* in literature. Moreover, in our formulation the (matrix) optimization variable X is binary. The *0-1 ILP* (also referred as *Binary ILP*, BILP) expressed as a decision problem constituted by only restrictions without an optimization function, has been classified as one of the *Karp's 21 NP-complete problems* [39].

Relaxing the elements of the X variable in the real domain, a convex problem could be obtained which could provide an useful upper bound for the optimal solution value of the original problem.

5.1. ADMM: unscaled form and augmented Lagrangian

The optimization problem, as expressed in the previous section, allows *CVX* and Gurobi to solve the problem, but it is still formulated in a completely centralized way. As stated above, such an approach can be hard to be applied in practice in the domain of our interest, since distributing resource characteristics and load of each node (with hundreds or thousands of nodes) is challenging. A possible way to solve the problem in a distributed fashion can be constituted by the iterative ADMM [20].

The original optimization problem is reported here:

$$\text{maximize} \sum_{j=0}^{N-1}\sum_{l=0}^{K-1} X(l,j), \tag{17}$$

subject to:

$$\mu X \leq X\nu, \tag{18a}$$

$$\sum_{j=0}^{N-1} X(l,j) \ \leq 1_K, \tag{18b}$$

$$execEnd_X \leq \tau_d. \tag{18c}$$

For ease of exposition, we rename the functions as:
$f(X) = -\sum_{j=0}^{N-1}\sum_{l=0}^{K-1} X(l,j)$, $\mu X = m_r(X)$ (the requested memory), $X\nu = m_a(X)$ (the memory available) and then $m_r(X) - m_a(X) = m(X)$ (memory constraint), $\sum_{j=0}^{N-1} X(l,j) = s(X)$ (number of executions / nodes for each task), $execEnd_X = e(X)$. Finally, the problem in Eqs. (17) and (18) can be rewritten as a minimization problem as follows:

$$\text{minimize} \quad f(X), \tag{19}$$

subject to:

$$m(X) \leq 0_K, \tag{20a}$$

$$s(X) \ \leq 1_K, \tag{20b}$$

$$e(X) \leq \tau_d. \tag{20c}$$

The ADMM (and more in general the augmented Lagrangian method) does not explicitly allow the use of inequality constraints, thus the use of a *slack vector* z for each constraint is required [40]. The inequality is implicitly considered in the optimization function requiring z to be non-negative through the indicator function $I_+(z)$ (that can be seen as an infinite penalty on the optimizing function when z is negative, and is zero otherwise). More precisely, the previous optimization problem is rewritten as:

$$\text{minimize} \quad f(X) + I_+(z_1) + I_+(z_2) + I_+(z_3), \tag{21}$$

subject to:

$$h_1(X, z_1) = m(X) + z_1 = 0, \tag{22a}$$

$$h_2(X, z_2) = s(X) - 1 + z_2 = 0, \tag{22b}$$

$$h_3(X, z_3) = e(X) - \tau_d + z_3. \tag{22c}$$

The augmented Lagrangian used in the ADMM can thus be easily written as:

$$L_\rho(X,z,y) = f(X) + \sum_{i=1}^{3} y_i^T h_i(X,z_i) + \sum_{i=1}^{3} \frac{\rho}{2}\|h_i(X,z_i)\|_2^2 + I_+(z_1) + I_+(z_2) + I_+(z_3), \tag{23}$$

and the ADMM (in unscaled form) is expressed as:

$$X^{k+1} = \underset{X}{\text{argmin}}\, L_\rho(X, z_i^k, y_i^k), \tag{24a}$$

$$z_i^{k+1} = \max\{\underset{z_i}{\text{argmin}}\, L_\rho(X^{k+1}, z_i, y_i^k),\ 0\}, \tag{24b}$$

$$y_i^{k+1} = y_i^k + \rho(h_i(X^{k+1}, z_i^{k+1})), \tag{24c}$$

with $i = 1, 2, 3$ and $\rho > 0$ (the augmented Lagrangian parameter).

More exactly, given that the problem is not convex, in the previous equation we should have considered $X^{k+1} \in \text{argmin}_X L_\rho(X, z_i^k, y_i^k)$, since $\text{argmin}_X L_\rho(X, z_i^k, y_i^k)$ is not necessarily a set made of only one element, but, for ease of presentation, we used the equality with the convention that X is any of the values in which the augmented Lagrangian attains its smallest value. A similar remark holds for Eq. (24b).

It is worth noting that the update of z_i in Eq. (24b) has required the use of the *max* operator to take into account that z_i cannot be negative (recalling the introduction of the I_+ function in the optimization problem to deal with the inequality constraints).

5.2. ADMM: scaled form and augmented Lagrangian

For ease of implementation the ADMM can be rewritten in scaled form. The terms in the augmented Lagrangian containing $r_i = h_i(X, z_i)$ can be transformed as:

$$\sum_{i=1}^{3} y_i^T r_i + \sum_{i=1}^{3} \frac{\rho}{2}\|r_i\|_2^2 = \sum_{i=1}^{3} \frac{\rho}{2}\|r_i + \frac{1}{\rho}y_i\|_2^2 - \sum_{i=1}^{3} \frac{1}{2\rho}\|y_i\|_2^2 \underset{u_i=\frac{1}{\rho}y_i}{=} = \sum_{i=1}^{3} \frac{\rho}{2}\|r_i + u_i\|_2^2 - \sum_{i=1}^{3} \frac{\rho}{2}\|u_i\|_2^2, \tag{25}$$

becoming:

$$L_\rho(X,z,u) = f(X) + I_+(z_1) + I_+(z_2) + I_+(z_3) + \sum_{i=1}^{3} \frac{\rho}{2}\|h_i(X,z_i) + u_i\|_2^2 - \sum_{i=1}^{3} \frac{\rho}{2}\|u_i\|_2^2, \tag{26}$$

and finally the ADMM expressed in scaled form is:

$$X^{k+1} = \underset{X}{\operatorname{argmin}} \left(f(X) + \sum_{i=1}^{3} \frac{\rho}{2} \| h_i(X, z_i) + u_i^k \|_2^2 \right), \tag{27a}$$

$$z_i^{k+1} = \max \left\{ \underset{z_i}{\operatorname{argmin}} \left(\sum_{i=1}^{3} \frac{\rho}{2} \| h_i(X, z_i) + u_i^k \|_2^2 \right), 0 \right\}, \tag{27b}$$

$$u_i^{k+1} = u_i^k + h_i(X^{k+1}, z_i^{k+1}). \tag{27c}$$

The scaled form makes easier the implementation in *Matlab* expressing the ADMM in a more compact and convenient form. Obviously, the two forms (scaled and unscaled) are equivalent.

5.3. Distributed ADMM using global consensus

The ADMM formulations defined in the previous Sections 5.1 and 5.2 allow only a marginal parallelization of the algorithm on z_i and y_i (u_i), where $i = \{1, 2, 3\}$. Reasoning about a real implementation of one of these formulations in the volunteer cloud environment, the ADMM turns out to be parallelized in only three computational threads but definitely not distributable on all the volunteers. To better exploit the distribution capabilities of the ADMM on all the volunteers, it is required to have X partitionable and f separable with respect to this partition. The original problem can be rewritten as a *Global Consensus* problem where the N nodes want to converge towards an optimal shared solution X. Expressing as $f_j(X) = -\sum_{l=0}^{K-1} X(l, j)$ the number of tasks per node (the "minus" sign is used to express the optimization problem as a minimization one), the problem in Eqs. (21) and (22) has the form:

$$\text{minimize} \quad \sum_{j=0}^{N-1} f_j(X) + \sum_{i=1}^{3} I_+(z_i), \tag{28}$$

subject to:

$$h_i(X, z_i) = 0 \quad \forall i = 1, 2, 3, \tag{29}$$

that can be distributed over the nodes introducing the local variables X_j, one for each node. Each X_j is a *copy* of the original X, and is updated locally and independently by each node. The global consensus variable Z is added. For ease of presentation the range of the j index is shifted from 1 to N in the following:

$$\text{minimize} \quad \sum_{j=1}^{N} f_j(X_j) + \sum_{i=1}^{3} I_+(z_i), \tag{30}$$

subject to:

$$X_j - Z = 0 \quad \forall j = 1, \ldots, N, \tag{31a}$$

$$h_i(Z, z_i) = 0 \quad \forall i = 1, 2, 3. \tag{31b}$$

The augmented Lagrangian is:

$$\begin{aligned} &L_\rho(X_1, \ldots, X_j, \ldots, X_N, Z, z_1, z_2, z_3, Y_1, \ldots, Y_N, y_{N+1}, y_{N+2}, y_{N+3}) \\ &\quad = \sum_{j=1}^{N} \left(f_j(X_j) + Y_j(X_j - Z) + \frac{\rho}{2} \| X_j - Z \|_2^2 \right) \\ &\quad + \sum_{i=1}^{3} \left(y_{N+i} h_i(Z, z_i) + \frac{\rho}{2} \| h_i(Z, z_i) \|_2^2 + I_+(z_i) \right). \end{aligned} \tag{32}$$

The ADMM for the global consensus problem becomes:

$$X_j^{k+1} = \underset{X_j}{\operatorname{argmin}} \left(f_j(X_j) + Y_j^k (X_j - Z^k) + \frac{\rho}{2} \| X_j - Z^k \|_2^2 \right) \quad \forall j = 1, \ldots, N, \tag{33a}$$

$$\begin{aligned} Z^{k+1} = \underset{Z}{\operatorname{argmin}} \Bigg(&\sum_{j=1}^{N} \left(- Y_j^k Z + \frac{\rho}{2} \| X_j^{k+1} - Z \|_2^2 \right) \\ &+ \sum_{i=1}^{3} \left(y_{i+N}^k h_i(Z, z_i^k) + \frac{\rho}{2} \| h_i(Z, z_i^k) \|_2^2 \right) \Bigg), \end{aligned} \tag{33b}$$

$$z_i^{k+1} = \max \left\{ \underset{z_i}{\operatorname{argmin}} \left(y_i^k h_i(Z^{k+1}, z_i) + \frac{\rho}{2} \| h_i(Z^{k+1}, z_i) \|_2^2 \right), 0 \right\} \quad \forall i = 1, 2, 3, \tag{33c}$$

$$Y_j^{k+1} = Y_j^k + \rho (X_j^{k+1} - Z^{k+1}) \quad \forall j = 1, \ldots, N, \tag{33d}$$

$$y_i^{k+1} = y_i^k + \rho (h_{i-N}(Z^{k+1}, z_i^{k+1})) \quad \forall i = N+1, N+2, N+3. \tag{33e}$$

Thus, the steps for updating X_j and Y_j can be executed locally on each node, while Z performs the role of the central collector for the optimization problems solved locally. Summing up, the ADMM formulation of the problem expressed in this section, compared to the solution proposed in the previous section, is characterized by an increase in the computation performed locally by the nodes. Some steps of the algorithm still need to be executed in a centralized fashion (i.e., the updates of the variables Z, z_i, y_i), while others could be executed locally (i.e., the updates of the variables X_j, Y_j).

5.4. Distributed ADMM using global consensus, scaled form

Similarly to what has been done in Section 5.2, substituting $R_j = X_j - Z$, $r_i = h_i(Z, z_i)$ and $u_i = \frac{1}{\rho} y_i$, $U_j = \frac{1}{\rho} Y_j$, the augmented Lagrangian can be rewritten as:

$$\begin{aligned} &L_\rho(X_1, \ldots, X_j, \ldots, X_N, Z, z_1, z_2, z_3, U_1, \ldots, U_N, u_{N+1}, u_{N+2}, u_{N+3}) \\ &\quad = \sum_{j=1}^{N} \left(f_j(X_j) + \frac{\rho}{2} \| R_j + U_j \|_2^2 - \frac{\rho}{2} \| U_j \|_2^2 \right) \\ &\quad + \sum_{i=1}^{3} \left(\frac{\rho}{2} \| r_i + u_{i+N} \|_2^2 - \frac{\rho}{2} \| u_{i+N} \|_2^2 + I_+(z_i) \right). \end{aligned} \tag{34}$$

Then, the scaled ADMM for the global consensus problem becomes:

$$X_j^{k+1} = \underset{X_j}{\operatorname{argmin}} \left(f_j(X_j) + \frac{\rho}{2} \| X_j - Z^k + U_j^k \|_2^2 \right) \quad \forall j = 1, \ldots, N, \tag{35a}$$

$$\begin{aligned} Z^{k+1} = \underset{Z}{\operatorname{argmin}} \Bigg(&\sum_{j=1}^{N} \left(\frac{\rho}{2} \| X_j^{k+1} - Z + U_j^k \|_2^2 \right) \\ &+ \sum_{i=1}^{3} \left(\frac{\rho}{2} \| h_i(Z, z_i^k) + u_{i+N}^k \|_2^2 \right) \Bigg), \end{aligned} \tag{35b}$$

$$z_i^{k+1} = \max \left\{ \underset{z_i}{\operatorname{argmin}} \left(\frac{\rho}{2} \| h_i(Z^{k+1}, z_i) + u_{i+N}^k \|_2^2 \right), 0 \right\} \quad \forall i = 1, 2, 3, \tag{35c}$$

$$U_j^{k+1} = U_j^k + X_j^{k+1} - Z^{k+1} \quad \forall j = 1, \ldots, N, \tag{35d}$$

$$u_i^{k+1} = u_i^k + h_{i-N}(Z^{k+1}, z_i^{k+1}) \quad \forall i = N+1, N+2, N+3. \tag{35e}$$

The fairness and distributed capabilities of this approach (problem formulated as global consensus and solved with ADMM) relies on the fact that each node can take its decision autonomously without a central decision system. Conversely, this approach compared to Section 5.2 has incremented the number of the constraints: from the initial 3 up to $N+3$ with the distributed (global consensus) approach.

5.5. A more *distributed ADMM using global consensus*

Observing the global consensus problem in Eq. (30) and the ADMM steps in Eqs. (33), it is worth noting that each y_i depends on the constraint h_i, which in turn depends on the global variable Z. This dependency does not allow to execute the step to update y_i in a distributed fashion. It is thus possible to define an equivalent problem considering the X_j local on each node even concerning the constraints h_i. Using X_j even for the constraints it is thus possible to increase the degree of distribution of the ADMM, applying it to the following equivalent optimization problem:

$$\text{minimize} \quad \sum_{j=1}^{N} f_j(X_j) + \sum_{i=1}^{3} I_+(z_i), \tag{36}$$

subject to:

$$X_j - Z = 0 \quad \forall j = 1, \ldots, N, \tag{37a}$$

$$h_i(X_j, z_i) = 0 \quad \forall i = 1, 2, 3 \wedge j = 1, \ldots, N. \tag{37b}$$

Thus the original constraints are distributed on each node becoming a total of $3 \cdot N$. In this way one obtains the augmented Lagrangian:

$$\begin{aligned} &L_\rho(X_1, \ldots, X_j, \ldots, X_N, Z, z_1, z_2, z_3, Y_1, \ldots, Y_j, \ldots, Y_N, y_{11}, \ldots, y_{ij}, \ldots) \\ &= \sum_{j=1}^{N} \left(f_j(X_j) + Y_j(X_j - Z) + \frac{\rho}{2}\|X_j - Z\|_2^2 \right) \\ &+ \sum_{j=1}^{N} \left(\sum_{i=1}^{3} \left(y_{ij} h_i(X_j, Z_i) + \frac{\rho}{2}\|h_i(X_j, z_i)\|_2^2 + I_+(z_i) \right) \right) \\ &= \sum_{j=1}^{N} \Bigg(f_j(X_j) + Y_j(X_j - Z) + \frac{\rho}{2}\|X_j - Z\|_2^2 \\ &+ \sum_{i=1}^{3} \left(y_{ij} h_i(X_j, Z_i) + \frac{\rho}{2}\|h_i(X_j, z_i)\|_2^2 + I_+(z_i) \right) \Bigg), \end{aligned} \tag{38}$$

and then, the ADMM steps (where, through the y_{ij}, each node j can manage its own subset i of constraints):

$$\begin{aligned} X_j^{k+1} = \underset{X_j}{\operatorname{argmin}} \Bigg(& f_j(X_j) + Y_j^k(X_j - Z^k) + \frac{\rho}{2}\|X_j - Z^k\|_2^2 \\ & + \sum_{i=1}^{3} \left(y_{ij}^k h_i(X_j, Z_i^k) + \frac{\rho}{2}\|h_i(X_j, Z_i^k)\|_2^2 \right) \Bigg) \quad \forall j = 1, \ldots, N, \end{aligned} \tag{39a}$$

$$Z^{k+1} = \underset{Z}{\operatorname{argmin}} \left(\sum_{j=1}^{N} \left(-Y_j^k Z + \frac{\rho}{2}\|X_j^{k+1} - Z\|_2^2 \right) \right), \tag{39b}$$

$$\begin{aligned} z_i^{k+1} = \max \left\{ \underset{z_i}{\operatorname{argmin}} \left(\sum_{j=1}^{N} \left(y_{ij}^k h_i(X_j^{k+1}, z_i) + \frac{\rho}{2}\|h_i(X_j^{k+1}, z_i)\|_2^2 \right), 0 \right) \right\} \\ \forall i = 1, 2, 3, \end{aligned} \tag{39c}$$

$$Y_j^{k+1} = Y_j^k + \rho(X_j^{k+1} - Z^{k+1}) \quad \forall j = 1, \ldots, N, \tag{39d}$$

$$y_{ij}^{k+1} = y_{ij}^k + \rho\left(h_i(X_j^{k+1}, z_i^{k+1})\right) \quad \forall i = 1, 2, 3 \wedge j = 1, \ldots, N. \tag{39e}$$

This new version of the distributed ADMM has more steps that can be executed in parallel (i.e., the updates of the y_{ij}) than the approach presented in Section 5.3.

5.6. About distributing the optimization problem with ADMM

Despite the ADMM does not allow to obtain a fully distributed approach, due to the presence of the *central collector*, great part of the problem is solved locally on the nodes, while the centralized effort is significantly reduced. In Table 1 we report the steps and the parallelism achievable with the three formulations of the ADMM. The steps identify the *barriers* (synchronization points) required by the parallelization, namely, the points in time where one variable depends on one or more variables computed in previous steps.

In the two distributed ADMM approaches described above (see Sections 5.3 and 5.5), for each iteration of the algorithm it is required to exchange a total of $2N$ messages among the nodes: N to distribute the local computation of X_j^{k+1} and Y_j^k to the central collector (*distribution phase*), and N to receive from the central collector the computation of Z^k (*gathering phase*).

It is worth noting that it could be possible to further parallelize the algorithm in Section 5.5 acting on z_i and introducing separate variables z_{ij} for each node, similarly to what has been done for y_{ij} in Section 5.5.

6. Implementation and numerical results

The methods have been implemented in *Matlab* with the *CVX toolbox* for the definition of the optimization problems and using the *Gurobi solver* which allows one to use integer variables.[1] This section presents the evaluated scenario (Section 6.1) and some numerical results (Section 6.2) comparing the two different versions of the ADMM (presented in Sections 5.2 and 5.4) with the initial ILP formulation solved in a centralized way (see the beginning of Section 5) and with two well-known scheduling algorithms, namely `Random-First Fit (Random-FF)` and `Round Robin-First Fit (RR-FF)` [41]. In `Random` and `RR`, task execution requests are forwarded to a node chosen, respectively, according to a uniform distribution and in circular order. The `First Fit` is an algorithm variant, usually considered in memory management contexts, consisting in stopping the search for a node once one that can satisfy the task request is found. An additional stopping criterion is usually associated to both the algorithms (i.e., `Random` and `RR`), in case none of the inquired nodes accepts the task execution. Otherwise both the approaches will turn into a *brute-force search* solving the original combinatorial optimization problem. For our experiments we arbitrarily chose to limit, for each task, the number of request to the 20% of the nodes involved in a given scenario. E.g., adopting the random scheduling and having 100 nodes, up to 20 nodes can be randomly selected and inquired for accepting a given task before stating that the task can not be executed. It is worth noting that the plain implementation of the `RR` algorithm requires a centralized coordination to be executed.

The centralized resolution of the ILP formulation provides the global maximum, while, as shown later in this section, the ADMM formulations usually provide good suboptimal solutions.

6.1. Scenario

We focus on two main aspects: the network and the workload models. As a matter of fact each simulation run has two stages:

[1] https://github.com/distributedResearch/taskSchedulingADMM.

Table 1
Parallelism capabilities of the ADMM formulations.

Step	ADMM centralized		ADMM distributed			ADMM more distributed		
	Variable	Parallelism	Variable	Parallelism		Variable	Parallelism	
I	X	1	X_j	N		X_j	N	
II	z_i	3	Z	1		Z	1	4
						z_i	3	
III	y_i	3	z_i	3	$3+N$	Y_j	N	$N+3\cdot N$
			Y_j	N		y_{ij}	$3\cdot N$	
IV			y_i	3				

Table 2
Node attributes.

CPU frequency (GHz)	CPU (cores)	RAM (GBs)
1 – 3	1 – 4	0.5 – 8

one for generating the network configuration (cloud participants), and another one for evaluating the actual activity period.

The nodes characteristics are shown in Table 2, where values are uniformly distributed within the specified intervals. We consider the communication overhead for transferring data as negligible. Since our problem formulation assumes a task distribution through allocation periods, the dynamism in the online presence of the volunteer nodes can be managed similarly. Evaluating allocation periods with a reasonable short duration, the optimization problem, in a first approximation, could consider only the nodes actually online at the beginning of such period. If a node correctly leaves the network (informing the other participants of its action), the tasks in its queue could be evaluated for re-execution in the subsequent allocation period. While, if a node abruptly leaves the network, all the tasks in its execution queue are lost.

For the workload characterization, our main reference is the Google Cloud Backend described in [42]. There, tasks are characterized according to their duration, CPU and memory requirements, each abstracted as either *small* (s) or *large* (l). For confidentiality reasons, the Google Cluster dataset [43] provides obfuscated information about the real hardware characteristics of the Google cluster nodes: every reported value is normalized to the capacity of the best cluster node. Another obfuscated information relevant for the purpose of our work regards the QoS properties such as deadlines. For these reasons we have made some assumptions. We consider that the CPUs of the cluster have a frequency of 1 GHz. In this work we have considered only the tasks of type *large*, whose attributes are uniformly distributed within the intervals (grouped by qualitative coordinates), as reported in Table 3.

Evaluating the task execution time as defined in Eq. (4), while using the characteristics of nodes and tasks considered during our experiments and defined above, it is possible to plot the time required to execute the tasks. In Fig. 2, the x and y axes represent, respectively, the execution capability that the node can exploit and the task duration while the z-axis shows the required execution time. From the plot it is possible noting that the task duration is largely affected by the node capabilities. In fact observing Table 3 the considered tasks have a good degree of parallelism that can be exploited to significantly reduce the required execution time (the blue region in the plot).

6.2. Results

In this section, we compare the optimal solution value of the original ILP optimization problem with values obtained applying the two versions of ADMM discussed in Section 5, and with `Random-FF` and `RR-FF` (discussed earlier in this section). In particular, the quality of the ADMM solutions has been evaluated. Since the ADMM, in presence of nonconvex problem, should be considered as a heuristic without any guarantee on convergence, we assume as stopping criterion 1000 iterations of ADMM.

All the experiments have been performed on a machine running Linux 3.16.0-46 64-bit equipped with an Intel Core i5-4460 and 32 GB of RAM, using *Matlab* 2015a with *CVX* 2.1 and the *Gurobi* solver version 6.0.4. A single run, with the largest problem, considering that our *Matlab* implementation is not able to exploit the parallelism of the algorithm, has required around one hour of execution to perform 1000 iterations of the ADMM (recalling that no other stopping criterion has been adopted). Increasing the parallelism, the subproblems for the computation of the ADMM variables become easier, and so the computational time for each iteration decreases.

To evaluate the scalability of the approach, we have considered different sizes of the problem (number of elements of X) varying K (number of tasks) and N (number of nodes). The results shown in this section are the averages of 10 runs, where the random seed changes the characteristics of nodes and tasks at each run according to Tables 2 and 3. *CVX* currently supports three solvers for problems that make use of integer variables, namely, *Gurobi, MOSEK and GLPK*. Unfortunately, none of these solvers compatible with *CVX* allows obtaining all the solutions in a straightforward way (it is only possible to apply some tricks to force the solver to skip the solutions already found, e.g., applying cuts that make the solutions already found unfeasible). Thus, comparing the approaches, not all the solutions of the problem found by the two ADMM implementations are identical. The main metric of interest during our experiments is the *hit rate*, defined as the number of tasks successfully executed (i.e., the value found for the optimization function normalized over K). This is evaluated at the best solution found by each method.

Performance results are summarized through descriptive statistics [44] (with empirical mean, median, 1st and 3rd quartile, min, max, standard deviation, and standard deviation of the estimated mean also called standard error of the mean) in Tables 4 and 5 and pictorially represented in Fig. 3 and, with box-plots, in Figs. 4 and 5.

Fig. 3 shows the numerical results of our comparisons. Increasing the size of the problem, the relative number of tasks that can be executed remains almost constant to the 80% (scaling X, the ratio between K and N is kept constant at about 1: 2). The global maximum (gray line), the performance of `Random-FF` and `RR-FF`, and the best value found by the ADMM formulations for the hit rate are shown in Fig. 3 a. The relative error, defined as the relative difference of each solution with the ILP one, is reported in Fig. 3 b. It is worth noting that, for each size of the problem, the ADMM formulations have found the global maximum in several runs. This observation is supported by the content of the rows referring to the minimum relative error and to the 1st quartile in Table 5, and

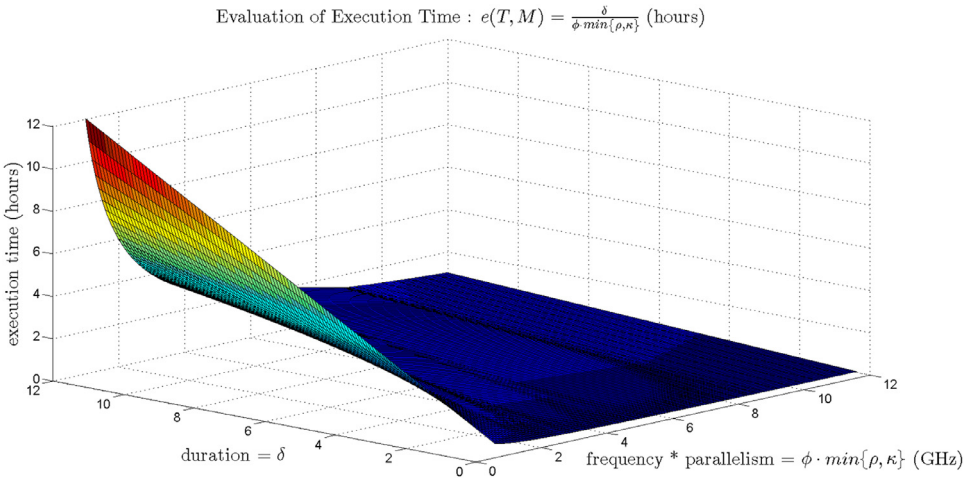


Fig. 2. Required execution time with varying tasks/nodes characteristics. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article).

Table 3
Task attributes.

Size	Duration (h)	CPU (cores)	RAM (GBs)	Deadline offset (% on duration)	Poisson mean arrival (ms)
Large	1 – 12	1 – 4	1 – 4	0.4	600

Table 4
Hit rate by problem complexity – descriptive statistics.

Algorithm	Metrics	Problem complexity			
		44	60	78	98
ILP	Mean	80	76	76.667	77.143
	Median	75	80	83.333	71.429
	1st quartile (25th percentile)	75	60	60.667	71.429
	3rd quartile (75th percentile)	100	80	83.333	85.714
	Min	50	60	66.667	71.429
	Max	100	100	83.333	85.714
	Stdev	18.708	14.967	8.165	6.999
	Std err of the mean	8.367	6.693	3.651	3.130
ADMM	Mean	55	56	60	62.857
	Median	50	60	66.667	71.429
	1st quartile (25th percentile)	50	60	50	57.143
	3rd quartile (75th percentile)	50	60	66.667	71.429
	Min	50	40	33.333	28.571
	Max	75	60	83.333	85.714
	Stdev	10	8	16.997	19.378
	Std err of the mean	4.472	3.578	7.601	8.667
distributed ADMM	Mean	60	56	63.333	65.714
	Median	50	60	66.667	57.143
	1st quartile (25th percentile)	50	60	66.667	57.143
	3rd quartile (75th percentile)	75	60	66.667	71.429
	Min	50	40	33.333	57.143
	Max	75	60	83.333	85.714
	Stdev	12.247	8	16.330	11.429
	Std err of the mean	5.478	3.578	7.303	5.111
FF-Random	Mean	35	36	43.333	37.143
	Median	25	40	50	42.857
	1st quartile (25th percentile)	25	20	33.333	28.571
	3rd quartile (75th percentile)	50	40	50	42.857
	Min	0	20	33.333	28.571
	Max	75	60	50	42.857
	Stdev	25.495	14.967	8.165	6.999
	Std err of the mean	11.402	6.693	3.651	3.130
FF-RR	Mean	65	52	60	45.714
	Median	75	40	66.667	42.857
	1st quartile (25th percentile)	50	40	50	28.571
	3rd quartile (75th percentile)	75	60	66.667	57.143
	Min	50	40	33.333	28.571
	Max	75	80	83.333	71.429
	Stdev	12.247	16	16.997	16.660
	Std err of the mean	5.477	7.155	7.601	7.451

Table 5
Relative error by problem complexity – descriptive statistics.

Algorithm	Metrics	Problem complexity			
		44	60	78	98
ADMM	Mean	0.267	0.23	0.2	0.187
	Median	0.333	0.25	0	0
	1st quartile (25th percentile)	0	0	0	0
	3rd quartile (75th percentile)	0.5	0.4	0.4	0.333
	Min	0	0	0	0
	Max	0.5	0.5	0.6	0.6
	Stdev	0.226	0.203	0.253	0.244
	Std err of the mean	0.101	0.091	0.113	0.109
distributed ADMM	Mean	0.2	0.23	0.16	0.147
	Median	0	0.25	0	0.2
	1st quartile (25th percentile)	0	0	0	0
	3rd quartile (75th percentile)	0.5	0.4	0.2	0.2
	Min	0	0	0	0
	Max	0.5	0.5	0.6	0.333
	Stdev	0.245	0.204	0.233	0.129
	Std err of the mean	0.110	0.091	0.104	0.058
FF-Random	Mean	0.583	0.58	0.43	0.52
	Median	0.5	0.5	0.4	0.5
	1st quartile (25th percentile)	0.5	0.4	0.4	0.5
	3rd quartile (75th percentile)	0.667	0.667	0.5	0.6
	Min	0.25	0.333	0.25	0.4
	Max	1	0.75	0.6	0.6
	Stdev	0.247	0.157	0.117	0.075
	Std err of the mean	0.111	0.070	0.0052	0.033
FF-RR	Mean	0.15	0.323	0.22	0.42
	Median	0	0.333	0.2	0.4
	1st quartile (25th percentile)	0	0.25	0	0.333
	3rd quartile (75th percentile)	0.25	0.333	0.4	0.6
	Min	0	0.2	0	0.167
	Max	0.5	0.5	0.5	0.6
	Stdev	0.2	0.102	0.204	0.166
	Std err of the mean	0.089	0.457	0.091	0.074

is pictorially represented by the box plots in Fig. 5 where, both the ADMM implementations, in all the studied problem complexity have the 1st quartile coinciding with 0. The centralized RR-FF is able to tie the performance of ADMM in some scenarios, but its performances are not stable varying the problem complexity, and it has a hit rate about 20% lower than ADMM in the most complex scenario (see the mean hit rate row in Table 4). Instead, the Random-FF performs worse for all of the considered problem complexity.

The box plots in Fig. 4, grouped by problem complexity, show some statistics of the hit rate results, in particular, through their quartiles. Some of the boxes referring to ADMM are collapsed in a single line since the 1*swe*st and 3rd quartiles overlap with the mean value.

Despite the ADMM is not always able to find the global optimum, in relation to the goodness of the local maximum and considering its ability to fairly distribute the execution on all the volunteer nodes, it should be considered a compelling approach in a real distributed implementation, when the centralized approach can not be practically applied.

It is worth observing that the size of the optimization variable could be reduced through a simple *pre-filtering* on the nodes' memory. Indeed, the memory is a hard-constraint (Eq. (12a)) that must be satisfied by each node independently from the choices of all other nodes and from the other accepted tasks. Thus, it is possible that, for each optimization period, all the nodes perform an initial bid with the set of tasks for which they can satisfy the memory constraint. The original problem can then be subdivided into simpler problems, where the memory constraint can be removed.

7. The optimization framework

In this section, we briefly formulate other task distribution policies which can be modeled according to our framework. The definition of these policies has proved to be straightforward and only minor changes in the objective function have been required while the ADMM implementation can be basically used *as-is*, changing the $f(X)$ in Section 5, as described in the following. The implementation of all the following policies could be used to obtain an upper bound for the global optimum while evaluating our other simulation-based studies e.g., [3,10,13,16,19].

7.1. Minimization of the termination time for the task set

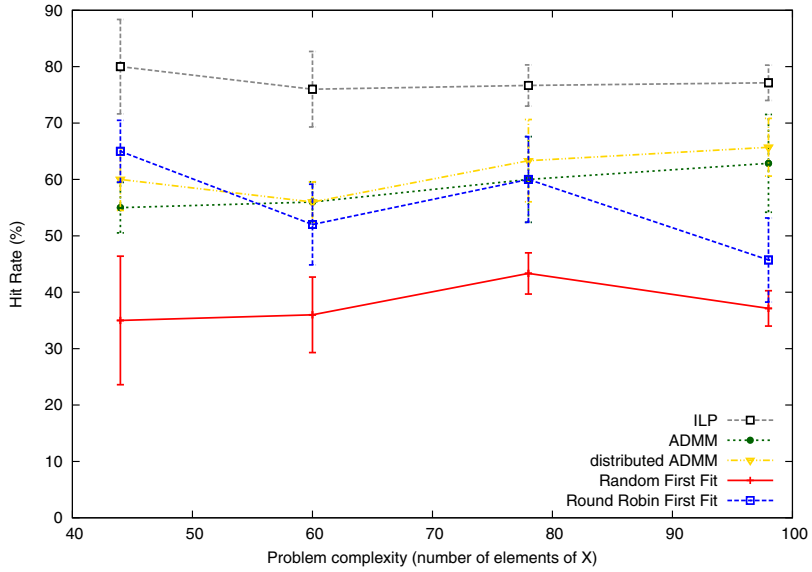
In this scenario, we want to minimize the time needed to complete all the tasks in the set. With this approach, the nodes become sooner free, and thus can more easily execute future requests in the subsequent allocation period.

The first step is the construction of the $execEnd_X$ vector as described in Eq. (12c), according to the actual choice of the executor nodes. Since the execution end time is zero for the tasks that are not executed, a trivial solution could be constituted by no task executed by any node. This misleading behavior can be eliminated stimulating the execution by adding a penalty factor for the tasks that are not actually executed:

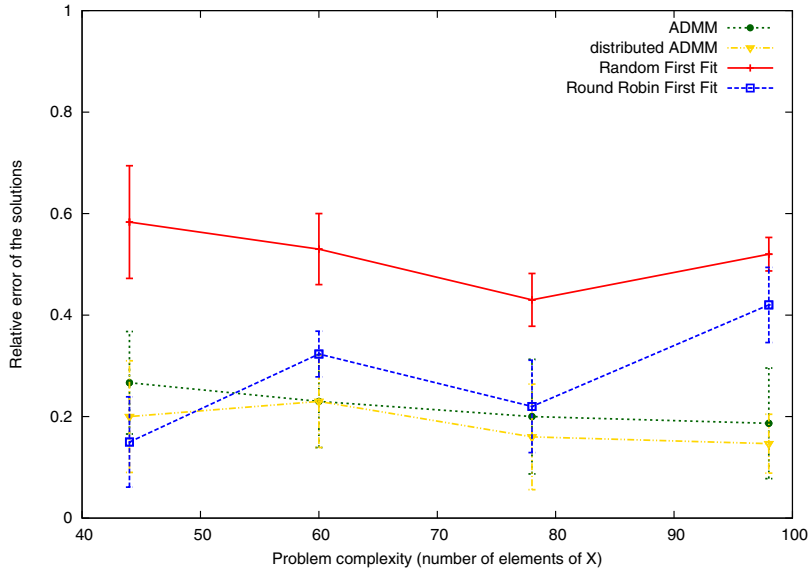
$$penalty := c \cdot \left(K - \sum_{j=0}^{N-1} executedTasks(j) \right), \tag{40}$$

where $c > 0$ is the penalty factor for not executing a task.

The optimization problem can then be expressed as the minimization of the sum of the total time needed to execute the tasks



(a) Hit rate %



(b) Relative error

Fig. 3. Numerical results for the ADMM, distributed ADMM, random-FF and RR-FF formulations: hit rate (top) and relative error (bottom). The vertical bar refers to the standard error of the mean.

and the penalty term mentioned above:

$$\text{minimize} \sum_{l=0}^{K-1} execEnd_X(l) + penalty, \tag{41}$$

subject to the constraints (a), (c) and (d).

In the penalty factor assignment, a more sophisticated strategy could take into account the class s of "importance" for the tasks that are not executed. This point will be considered in a future work.

7.2. Minimization of the response time

The minimization of the response time proceeds similarly to the minimization of the termination time for the task set (Section 7.1). The main difference relies on the choice of the constraints. In this problem, each task is assigned to a node, even if is not possible to satisfy the deadline requirement. This constraint makes useless the inclusion of a penalty factor and thus the problem is formulated as:

$$\text{minimize} \sum_{l=0}^{K-1} execEnd_X(l), \tag{42}$$

subject to the constraints (a) and (b).

7.3. Maximization of nodes fairness

In this case, the goal is the maximization of the distribution of tasks among nodes. Thus, the highest number of nodes should participate in the execution of the task set. The objective function measures the *node_fairness* and is computed as the difference of tasks executed by the node that executes more tasks to the node that executes less:

$$\Delta_{task} = \max(executedTasks) - min(executedTasks). \tag{43}$$

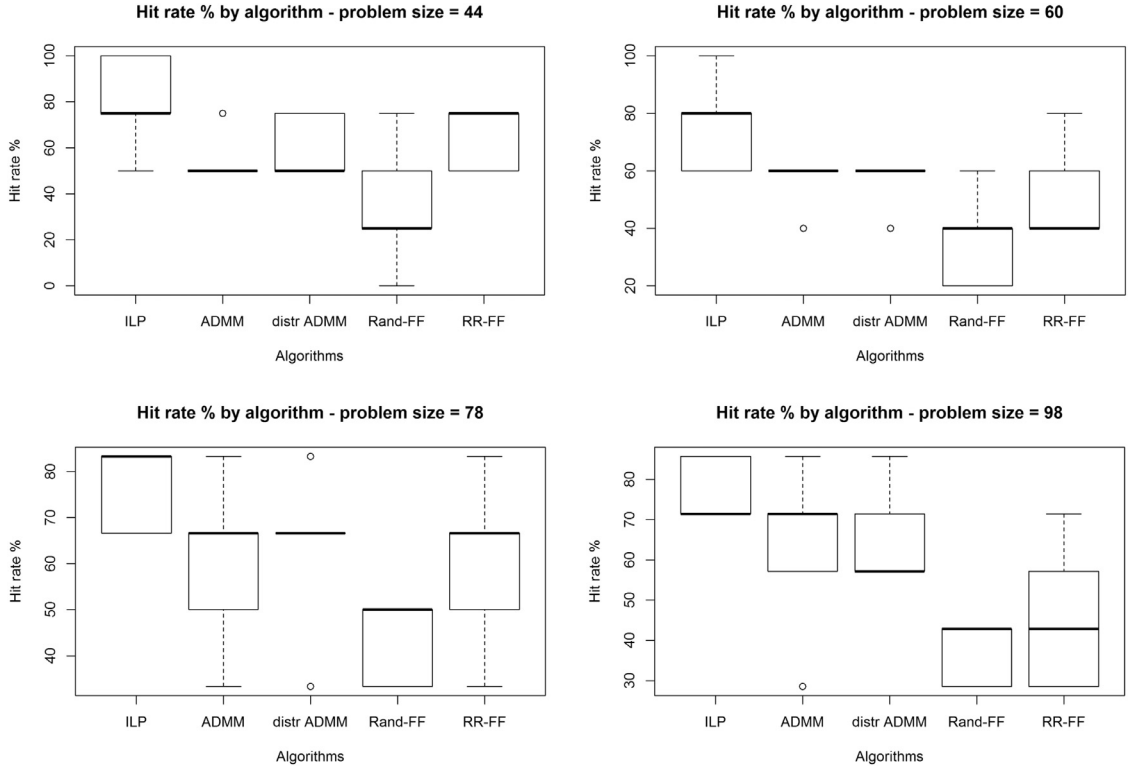


Fig. 4. Box plots for the hit rate, grouped by problem complexity (in increasing order).

This strategy, if implemented trivially, brings to a situation where none of the nodes execute tasks since this situation allows to have $\Delta_{task} = 0$. To prevent this vicious behavior from behalf of the nodes, the number of executed tasks is checked and for each not executed task a penalty is assigned (Eq. (40)). It is possible to express this problem as a maximization problem: $maximize(node\_fairness(\cdots))$, or dually as a minimization one: $minimize(node\_unfairness(\cdots)$. Our implementation follows the latter approach:

$$\text{minimize} \quad \Delta_{task} + penalty, \tag{44}$$

subject to the constraints (a), (c) and (d).

7.4. Maximization of nodes greenness

In this case, the goal is the reduction of the number of nodes performing computation in a given allocation period, while taking into account the nodes that are already active and running tasks belonging to previous periods. The underlying idea is that in this way the nodes that do not execute any task can shift towards an *energy-saving* state. The executed tasks must still respect the deadline constraints, but solutions that use a lower number of nodes are preferred.

This green policy consolidates the tasks in the smallest possible number of nodes but, to prevent that no tasks will be executed, a penalty factor is added (Eq. (40)). Maximizing the greenness (i.e., maximizing the number of nodes *off*) is analogous to minimizing the *green cost* (i.e., minimizing the number of nodes *on*). In the following we adopted the latter approach. Given $c_2 > 0$ and $b > 0$, the computation of the green cost is articulated in a few steps:

$$activeNodes_t = \min\left\{\sum_{l=0}^{K-1} X(l, j), 1_N\right\}, \tag{45a}$$

$$switched = activeNodes_{t-1} - activeNodes_t, \tag{45b}$$

$$greenCost = c_2 \cdot \sum_{j=0}^{N-1} |\ \min\{switched(j), 0_N\}| + - b \cdot \sum_{j=0}^{N-1} \max\{switched(j), 0_N\} + penalty. \tag{45c}$$

The active nodes are evaluated in Eq. (45a), where the *min* operator is used to count a node as active disregarding the number of tasks it is executing (since just one task is enough to require the node to be running). Then, in Eq. (45b) the nodes that should change their state in the current allocation period t are evaluated, i.e., each element of the vector $switched \in \mathbb{R}^N$ is 1 if a node can be switched to an energy-saving state, -1 if a new node needs to be activated, and 0 if no change of the state occurs. It follows that the undesired situations occur for the -1 elements of *switched*, while 1 represents the preferred value. Finally, the *greenCost* is computed in Eq. (45c), where the penalty factor c_2 takes into account the nodes that are required to wake-up, while b is a bonus for the nodes switched to energy-saving state.

In fact, the green optimization problem as formulated in Eqs. (45), turns out to be a multi-criteria optimization problem, where a trade-off among the benefit of executing more tasks (last terms of Eq. (45c)) and the use of fewer nodes (the middle term) should be weighted with the need to wake-up further nodes. The optimization problem considers the minimization of the *greenCost* (Eq. (45c)) and it is subject to the constraints (a), (c) and (d) of our framework.

Other policies to *greening* the volunteer cloud can be found in [45].

8. Conclusions and future work

The volunteer cloud computing is characterized by a large-scale heterogeneous and dynamic environment. Such a complex envi-

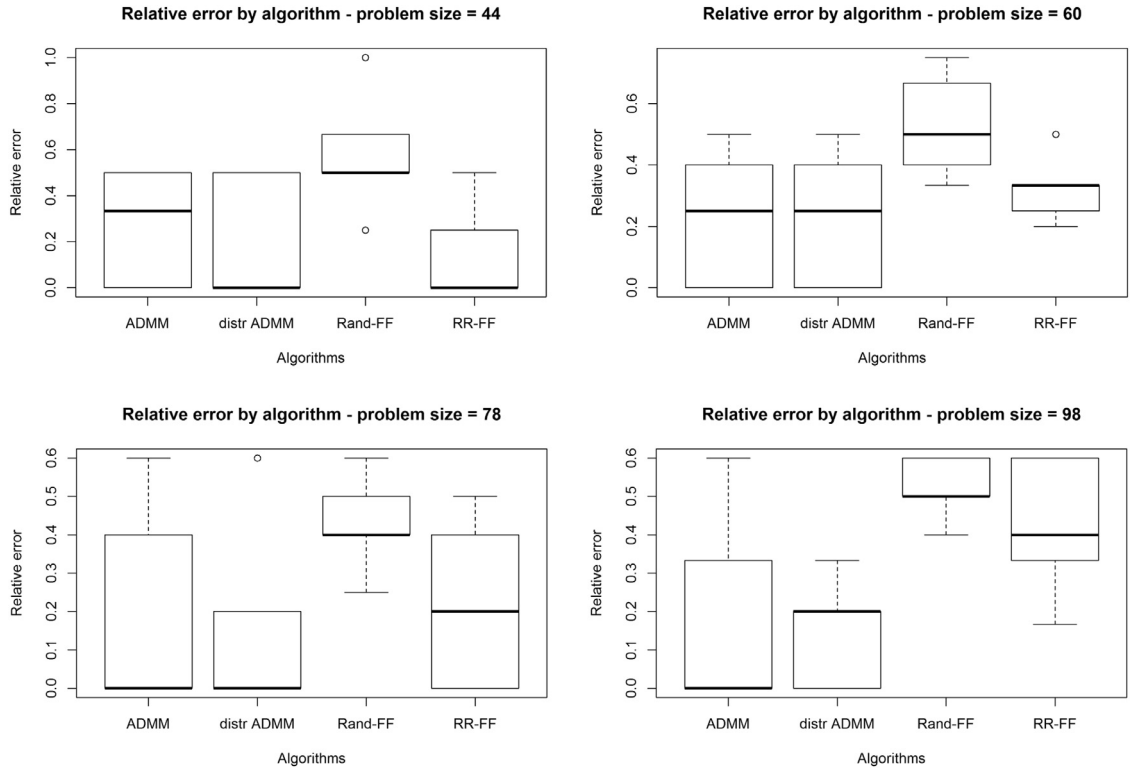


Fig. 5. Box plots for the relative error with respect to the global optimum solution (ILP), grouped by problem complexity (in increasing order).

ronment makes it hard, if not impossible, to perform global optimization algorithms, and to distribute tasks execution requests (often with associated deadline requirements), in a centralized fashion. Distributed algorithms are thus advocated, as the only solution methods applicable in a real environment.

In this work, we propose a distributed framework defining the optimization problem related to the allocation of the tasks execution requests according to different policies. Our problem formulation has been driven by the requirements of a real environment that we considered in our other previous simulative works [3,10,13,16,19] based on pure heuristic solutions. At the best of our knowledge, those requirements are instead often neglected in the literature related to the volunteer cloud, simplifying the domain specification while formalizing the problem as an optimization problem. Examples of the key elements of our modeling are: execution queue with FIFO policy, tasks with associated deadline, and evaluation of the actual load on the machines. Without loss of generality in the context of our framework, one of these policies has been implemented relying on the ADMM, with various parallelism capabilities. The numerical results show that, despite the ADMM is just a heuristic in presence of a non-convex problem (as in our formulation), it can still be used successfully. Up to the authors' knowledge, the application of ADMM for task scheduling in the context of the volunteer cloud is novel.

We conclude by mentioning that, in a real domain, a single policy could be driven by multiple goals. We are investigating how to fruitfully integrate *multi-criteria* optimization techniques [46] in our framework. A further point that can be investigated to increase the realism of our model is related to the time required to transfer the task towards the executor node (recalling that, in the volunteer cloud, each node acts as both task producer and consumer). E.g., evaluating the *execTime* it is possible to build a "local-view" for each node. In each allocation period the tasks could be sorted (preferred) by each node according to the above mentioned transmission cost.

Acknowledgment

This research was partially supported by the EU through the HOME/2013/CIPS/AG/4000005013 project CI2C. The contents of the paper do not necessarily reflect the position or the policy of funding parties.

References

[1] Cunsolo VD, Distefano S, Puliafito A, Scarpa M. Volunteer computing and desktop cloud: the cloud@home paradigm. In: Proceedings of the eighth IEEE international symposium on network computing and applications (NCA); 2009. p. 134–9. doi:10.1109/NCA.2009.41.

[2] Satyanarayanan M, Bahl P, Caceres R, Davies N. The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput 2009;8(4):14–23. doi:10.1109/MPRV.2009.82.

[3] Sebastio S, Amoretti M, Lluch Lafuente A. AVOCLOUDY: a simulator of volunteer clouds. Softw Pract Exp 2016;46(1):3–30. doi:10.1002/spe.2345.

[4] Costa F, Silva L, Dahlin M. Volunteer cloud computing: mapreduce over the internet. In: Proceedings of the 2011 IEEE international symposium on parallel and distributed processing workshops and Phd forum (IPDPSW); 2011. p. 1855–62. doi:10.1109/IPDPS.2011.345.

[5] Anderson DP. BOINC: a system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM international workshop on grid computing (GRID '04). Washington, DC, USA: IEEE Computer Society; 2004. p. 4–10. ISBN 0-7695-2256-4. doi:10.1109/GRID.2004.14.

[6] Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience.. Concurr Pract Exp 2005;17(2-4):323–56.

[7] Brasileiro F, Araujo E, Voorsluys W, Oliveira M, Figueiredo F. Bridging the high performance computing gap: the ourgrid experience. In: Proceedings of the seventh IEEE international symposium on cluster computing and the grid (CCGRID '07). Washington, DC, USA: IEEE Computer Society; 2007. p. 817–22. ISBN 0-7695-2833-3. doi:10.1109/CCGRID.2007.28.

[8] Cappos J, Beschastnikh I, Krishnamurthy A, Anderson T. Seattle: a platform for educational cloud Computing. SIGCSE Bull 2009;41(1):111–15. doi:10.1145/1539024.1508905.

[9] Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D. SETI@home: an experiment in public-resource computing. Commun ACM 2002;45(11):56–61. doi:10.1145/581571.581573.

[10] Sebastio S, Scala A. A workload-based approach to partition the volunteer cloud. In: Proceedings of the 2015 IEEE conference on collaboration and internet computing (CIC); 2015. p. 210–18. doi:10.1109/CIC.2015.27.
[11] Babaoglu O, Marzolla M, Tamburini M. Design and implementation of a P2P cloud system. In: Proceedings of the 27th annual ACM symposium on applied computing (SAC '12). New York, NY, USA: ACM; 2012. p. 412–17. ISBN 978-1-4503-0857-1. doi:10.1145/2245276.2245357.
[12] Di Nitto E, Dubois DJ, Mirandola R. On exploiting decentralized bio-inspired self-organization algorithms to develop real systems. In: Proceedings of the 2009 ICSE workshop on software engineering for adaptive and self-managing systems (SEAMS '09). Washington, DC, USA: IEEE Computer Society; 2009. p. 68–75. ISBN 978-1-4244-3724-5. doi:10.1109/SEAMS.2009.5069075.
[13] Amoretti M, Lafuente AL, Sebastio S. A cooperative approach for distributed task execution in autonomic clouds. In: Proceedings of the 2013 21st euromicro international conference on parallel, distributed and network-based processing, (PDP); 2013. p. 274–81. doi:10.1109/PDP.2013.47.
[14] Talia D. Cloud computing and software agents: towards cloud intelligent services. In: Fortino G, Garro A, Palopoli L, Russo W, Spezzano G, editors. Proceedings of the 12th workshop on objects and agents, Rende (CS), Italy, Jul 4-6, 2011. CEUR Workshop Proceedings, vol. 741; 2011. p. 2–6. CEUR-WS.org
[15] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1997;1(1):53–66.
[16] Sebastio S, Amoretti M, Lluch Lafuente A. A computational field framework for collaborative task execution in volunteer clouds. In: Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems (SEAMS 2014). New York, NY, USA: ACM; 2014. p. 105–14. ISBN 978-1-4503-2864-7. doi:10.1145/2593929.2593943.
[17] Tsai J-T, Fang J-C, Chou J-H. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Comput Oper Res 2013;40(12):3045–55. doi:10.1016/j.cor.2013.06.012.
[18] Zambonelli F., Mamei M. Spatial computing: an emerging paradigm for autonomic computing and communication. In: Smirnov M., editor. Autonomic communication; vol. 3457of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. ISBN 978-3-540-27417-9;, p. 44–57. 10.1007/11520184_4.
[19] Celestini A, Lluch Lafuente A, Mayer P, Sebastio S, Tiezzi F. Reputation-based cooperation in the clouds. In: Zhou J, Gal-Oz N, Zhang J, Gudes E, editors. Trust management VIII. IFIP Advances in Information and Communication Technology, vol. 430. Berlin, Heidelberg: Springer; 2014. p. 213–20. ISBN 978-3-662-43812-1. doi:10.1007/978-3-662-43813-8_15.
[20] Boyd S, Parikh N, Chu E, Peleato B, Eckstein J. Distributed optimization and statistical learning via the alternating direction method of multipliers. Found Trends Mach Learn 2011;3(1):1–122.
[21] Grant M., Boyd S.. CVX: matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx; 2014.
[22] Gurobi Optimization I.. Gurobi Optimizer Reference Manual. 2015. http://www.gurobi.com.
[23] Haridas H, Kailasam S, Dharanipragada J. Cloudy knapsack problems: an optimization model for distributed cloud-assisted systems. In: Proceedings of the 14th IEEE international conference on peer-to-peer computing (P2P); 2014. p. 1–5. doi:10.1109/P2P.2014.6934313.
[24] Malawski M, Figiela K, Nabrzyski J. Cost minimization for computational applications on hybrid cloud infrastructures. Futur Gener Comput Syst 2013;29(7):1786–94. doi:10.1016/j.future.2013.01.004.
[25] Fourer R, Gay DM, Kernighan B. AMPL: a modeling language for mathematical programming. 2nd. Cengage Learning; 2002. ISBN 0-534-38809-4.
[26] Malawski M, Figiela K, Bubak M, Deelman E, Nabrzyski J. Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. Sci Program 2015;2015:1–13. doi:10.1155/2015/680271.
[27] Wendell P, Jiang JW, Freedman MJ, Rexford J. DONAR: decentralized server selection for cloud services. SIGCOMM Comput Commun Rev 2010;41(4). doi:10.1145/1851275.1851211.
[28] Xu H, Li B. Joint request mapping and response routing for geo-distributed cloud services. In: Proceedings IEEE INFOCOM; 2013. p. 854–62. doi:10.1109/INFCOM.2013.6566873.
[29] Li B, Song SL, Bezakova I, Cameron KW. EDR: an energy-aware runtime load distribution system for data-intensive applications in the cloud. In: Proceedings of the 2013 IEEE international conference on cluster computing (CLUSTER); 2013. p. 1–8. doi:10.1109/CLUSTER.2013.6702674.
[30] Nedic A, Ozdaglar A, Parrilo PA. Constrained consensus and optimization in multi-agent networks. IEEE Trans Autom Control 2010;55(4):922–38. doi:10.1109/TAC.2010.2041686.
[31] Zhu Q, Zeng H, Zheng W, Natale MD, Sangiovanni-Vincentelli A. Optimization of task allocation and priority assignment in hard real-time distributed systems. ACM Trans Embed Comput Syst 2013;11(4):1–30. doi:10.1145/2362336.2362352.
[32] IBM. ILOG CPLEX optimizer. https://www.ibm.com/software/commerce/optimization/cplex-optimizer, March 2014.
[33] Boyd S, Vandenberghe L. Convex optimization. New York, NY, USA: Cambridge University Press; 2004. ISBN 0521833787.
[34] Ghadimi E, Teixeira A, Shames I, Johansson M. Optimal parameter selection for the alternating direction method of multipliers (ADMM): quadratic problems. IEEE Trans Autom Control 2015;60(3):644–58. doi:10.1109/TAC.2014.2354892.
[35] Feizollahi MJ, Costley M, Ahmed S, Grijalva S. Large-scale decentralized unit commitment. Int J Electr Power Energy Syst 2015;73(0):97–106. doi:10.1016/j.ijepes.2015.04.009.
[36] Miksik O, Vineet V, Pérez P, Torr P. Distributed non-convex ADMM-based inference in large-scale random fields. Proceedings of the British machine vision conference. BMVA Press; 2014.
[37] The service level agreement. 2015. http://www.sla-zone.co.uk.
[38] Grant M, Boyd S. Graph implementations for nonsmooth convex programs. In: Blondel V, Boyd S, Kimura H, editors. Recent advances in learning and control. Lecture Notes in Control and Information Sciences. Springer-Verlag Limited; 2008. p. 95–110. http://stanford.edu/~boyd/graph_dcp.html
[39] Karp RM. Reducibility among combinatorial problems. In: Miller R, Thatcher J, Bohlinger J, editors. Complexity of computer computations. The IBM Research Symposia Series. US: Springer; 1972. p. 85–103. ISBN 978-1-4684-2003-6. doi:10.1007/978-1-4684-2001-2_9.
[40] Nocedal J, Wright SJ. Numerical optimization. 2nd. New York: Springer; 2006.
[41] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 3rd. The MIT Press; 2009.
[42] Mishra AK, Hellerstein JL, Cirne W, Das CR. Towards characterizing cloud backend workloads: insights from google compute clusters. ACM SIGMETRICS Perform Eval Rev 2010;37(4):34–41.
[43] Hellerstein J.L.. Google cluster data, Google research blog, 2010. Posted at http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.
[44] Trivedi KS. Probability and statistics with reliability, queuing and computer science applications. 2nd ed. Chichester, UK: John Wiley and Sons Ltd.; 2002. ISBN 0-471-33341-7.
[45] Sebastio S., Gnecco G.. A green policy to schedule tasks in a distributed cloud. Submitted for publication 2016.
[46] Ehrgott M. Multicriteria optimization. 2nd ed. Springer; 2005. ISBN 978-3-540-21398-7. doi:10.1007/3-540-27659-9.