A Holistic Approach for Collaborative Workload Execution in Volunteer Clouds

STEFANO SEBASTIO, London Institute of Mathematical Sciences
MICHELE AMORETTI, Università degli Studi di Parma
ALBERTO LLUCH LAFUENTE, Technical University of Denmark
ANTONIO SCALA, Institute for Complex Systems–CNR

The demand for provisioning, using, and maintaining distributed computational resources is growing hand in hand with the quest for ubiquitous services. Centralized infrastructures such as cloud computing systems provide suitable solutions for many applications, but their scalability could be limited in some scenarios, such as in the case of latency-dependent applications. The *volunteer cloud* paradigm aims at overcoming this limitation by encouraging clients to offer their own spare, perhaps unused, computational resources. Volunteer clouds are thus complex, large-scale, dynamic systems that demand for self-adaptive capabilities to offer effective services, as well as modeling and analysis techniques to predict their behavior. In this article, we propose a novel holistic approach for volunteer clouds supporting collaborative task execution services able to improve the quality of service of compute-intensive workloads. We instantiate our approach by extending a recently proposed ant colony optimization algorithm for distributed task execution with a workload-based partitioning of the overlay network of the volunteer cloud. Finally, we evaluate our approach using simulation-based statistical analysis techniques on a workload benchmark provided by Google. Our results show that the proposed approach outperforms some traditional distributed task scheduling algorithms in the presence of compute-intensive workloads.

CCS Concepts: • **Theory of computation** → **Scheduling algorithms**; • **Computing methodologies** → **Multi-agent planning**; **Distributed artificial intelligence**; • **Computer systems organization** → *Distributed architectures*; • **Software and its engineering** → *Software architectures*;

Additional Key Words and Phrases: Collective adaptive systems, multiagent optimization, collaborative computing, task scheduling, cloud computing, ant colony optimization (ACO), autonomic computing, computational fields, peer-to-peer (P2P)

ACM Reference format:
Stefano Sebastio, Michele Amoretti, Alberto Lluch Lafuente, and Antonio Scala. 2018. A Holistic Approach for Collaborative Workload Execution in Volunteer Clouds. *ACM Trans. Model. Comput. Simul.* 28, 2, Article 14 (March 2018), 27 pages.
https://doi.org/10.1145/3155336

Authors' addresses: S. Sebastio, LIMS London Institute for Mathematical Sciences, 22 South Audley St Mayfair, London, W1K2XF, United Kingdom; email: stefano.sebastio@alumni.imtlucca.it; M. Amoretti, Department of Engineering and Architecture, University of Parma, Parco Area delle Scienze, 181A, 43124 Parma, Italy; email: michele.amoretti@unipr.it; A. L. Lafuente, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, Building 324, 2800 Kgs, Lyngby, Denmark; email: albl@dtu.dk; A. Scala, ISC-CNR Physics Department, University "La Sapienz", Piazzale Moro 5, 00185, Roma, Italy; email: antonio.scala@phys.uniroma1.it.

https://doi.org/10.1145/3155336

1 INTRODUCTION

The increasing role of software and the pervasiveness of computational devices have fostered the interest and demand for storage and computing services. Cloud computing is a recent but already well-established paradigm envisioning the use of *computing as utility*. Thanks to the cloud, users do not need to buy or maintain powerful computational devices, as they can use services remotely. Such services are customized according to the users' needs and requirements, and are provided transparently from where the resources are physically hosted. Another benefit of cloud systems is their ability to dynamically scale whenever application requirements change. To properly manage their resources, cloud providers offer contract-based services whose terms are specified in a service-level agreement (SLA), containing the required quality of service (QoS) and the associated penalty fees in case of their violation.

One of the greatest challenges in the cloud computing domain is the efficient use of resources while guaranteeing the fulfillment of SLAs. To this aim, particular attention is given to autonomic computing techniques, which support the development of clouds as self-managing systems. Cloud features should automatically adapt to their internal status and the dynamically changing environment, without human intervention. Self-management may involve system maintenance, awareness, evolution, configuration, healing, tuning, and optimization.

Along with the latest advancements in virtualization technologies, the past decade has also witnessed a great leap forward in processing power, thanks to the advent of the multicore era. In turn, this scenario has brought a surge of resources that are largely unused for great part of the day, such as when desktop and laptop devices are used only for Web-browsing or text-editing activities. However, these personal devices have scarce resources for other activities, such as large-scale simulations or graphic editing. The volunteer computing paradigm thrives in such a scenario, as it foresees the willingness of some users to cooperate by sharing a fraction of their spare and unused resources while accessing some other users' shared resources when the local ones are scarce. By doing so, the limits in the scalability of clouds are overcome, as now users themselves may provide new computational resources.

The volunteer cloud computing paradigm promises to enhance clouds in specific domains, by exploiting peer-to-peer (P2P) networks of volunteers. This paradigm is also referred to as the social cloud, P2P cloud computing, and distributed cloud computing. A typical example of its use is given by academic and industrial research institutions that share the spare resources of their labs (not used 24 hours a day or underused) with other sites, to run complex large-scale simulations. Volunteer clouds support both computing- and storage-oriented cloud-based applications. For example, MapReduce and streaming applications can be successfully implemented in a volunteer cloud fashion [12]. Instead, the volatile online presence of the volunteer nodes prevents its use for service-oriented applications such as multitier Web applications.

Several works [6, 24, 25, 34] have identified domains suitable for standing out the volunteer cloud's benefits and opportunities. Exploiting the unused computational resources made available free of charge reduces the cost to maintain private computational resources. Moreover, the requirements for building and maintaining huge power-hungry and cooling-demanding data centers could be reduced by moving toward a green computing vision. Finally, the lack of a single controlling entity makes volunteer clouds convenient in case of natural catastrophes that would damage data centers or communication infrastructures, or when governments place restrictions on the location of sensitive data, as well as reduce the risk in the vendor lock-in problem.

The participation of volunteers can be fostered by using incentives such as rewards for participants who respect the SLAs associated to the assigned tasks. For example, the PlanetLab platform [57] fosters participation in its P2P network, requiring members to share a minimum number

of machines (with specified minimum system requirements) to gain access to all other shared machines.

The increasing interest in the volunteer cloud paradigm [5] is also witnessed by various publicly funded projects-for example, in France at INRIA (Clouds@Home), in Italy at the University of Bologna (Peer-to-Peer Cloud System) and the University of Messina (Cloud@Home), and in general within the European Community (Nano Data Centers (NADA) [36]; the European Desktop Grid Initiative (EDGI) [17]); the CERN's volunteer cloud [51]; in the United States within the National Science Foundation (NSF), which supports the BOINC project (to date with five research grants [60]), HTCondor (to date with four grants [62]), Seattle (to date with three grants [8]), and SETI@home (with support from NSF and NASA [61]); and the Federal University of Campina Grande in Brasil with the OurGrid initiative. Other works have modeled resource discovery algorithms in volunteer clouds relying on queueing theory [19, 20] and explored the opportunity for distributing Matlab simulations in a university volunteer infrastructure [10]. Despite all of those research and implementation efforts, several challenges posed by such a large-scale, distributed, heterogeneous, and dynamic environment lie ahead. There is a need to develop architectures and models to engineer volunteer cloud computing systems and to predict their behavior.

Contribution. Our contribution to this field focuses on volunteer cloud computing systems providing collaborative task execution services. In such systems, participants can join and leave at any time, request the execution of tasks subject to certain QoS constraints, and contribute with part of their own computational resources to the collective task execution. We have been developing autonomic computing architectures [3], adaptive artificial intelligence algorithms to build and exploit distributed data structures called *colored computational fields* using ant colony optimization (ACO) [46], dynamic techniques to structure overlay networks and adapt them to the characteristics of the tasks to be executed [49], and discrete event simulators to evaluate the proposed solutions using simulation-based statistical analysis [45]. This article frames and extends our previous works in a comprehensive manner. All in all, this work provides the following contributions:

- We present a holistic approach to master the complexity of volunteer cloud systems. In this context, *holistic* means that all (relevant) layers (namely, the application execution and the overlay layer) of the architecture of all nodes cooperate to provide a collective service.
- We show the flexibility of the approach, which allows one to combine analysis and optimization capabilities acting at different layers in a variegated way, by framing the resource discovery and task scheduling technique from Sebastio et al. [46] and the overlay network shaping technique from Sebastio and Scala [49] in our approach, and we present their combination for the first time.
- We present simulation results showing that such a novel combination of techniques outperforms their sole application, as well other collaboration strategies in the presence of compute-intensive workloads.

One key point of our work lies in showing the benefits of using agents to *collectively* cooperate for performance optimization at all layers involved in a volunteer cloud instead of focusing on just one layer.

Synopsis. The rest of the article is structured as follows. Section 2 presents our holistic approach to volunteer clouds supporting task execution services. Section 3 illustrates the instantiation of the approach with the two adaptive optimization techniques taken from Sebastio et al. [46] and Sebastio and Scala [49]. A performance evaluation carried out through a simulation-based statistical analysis using workload data trace from Google is presented in Section 4. Section 5

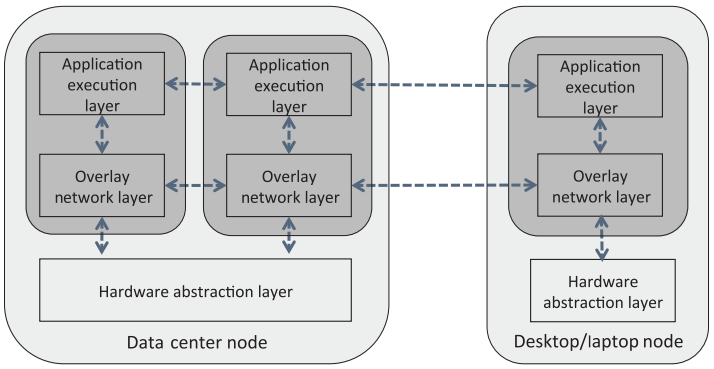


Fig. 1. Our architecture for volunteer clouds: main components and interactions.

discusses related work. Section 6 concludes the article, providing some final remarks and outlining our current and future research efforts.

2 A HOLISTIC APPROACH TO COLLABORATIVE VOLUNTEER CLOUDS

In accordance with the "cloud using agents" vision introduced by Talia [55], we proposed in Sebastio [44] a novel architecture for volunteer clouds. The main components of the architecture and the main interactions are illustrated in Figure 1. Each physical node (light gray rounded boxes) corresponds to a physical machine and may support more than one logical node (dark gray rounded boxes), such as in the case of powerful data centers. The three layers (square boxes) under consideration are the *hardware abstraction layer* (providing an abstraction of the underlying physical resources), the *overlay network layer* (in charge of basic P2P functionalities like node discovery), and the *application execution layer* (providing the task execution service through suitable scheduling strategies). Notably, the architecture is able to accommodate a dynamic resource sharing paradigm (in place of a persistent service-oriented provisioning provided by the typical cloud) with an on-the-fly remote relocation and execution of tasks. Each layer is coordinated by an agent acting according to the MAPE-K model [26] and interacts with other agents in the same layer but on different logical nodes, and with the agents managing the upper and lower layer within the same node. Such a layered and modular agent-based architecture enables the realization of the system's self-adaptive capabilities. Thanks to this architecture, which decouples the overlay network layer from the application execution layer, it has been possible to design a holistic approach in which all agents collaborate with the final goal of better supporting the task execution service.

The overlay network in P2P systems is usually in charge of providing communication and discovery services and ensuring its own resiliency (through health-checking and self-healing operations, i.e., the actions sustaining the network functionalities to ensure proper connectivity and to restore the P2P overlay in case of abrupt node disconnections without prior notification). In autonomic volunteer clouds, the overlay network can provide additional functionalities thanks to the use of intelligent agents. In our vision, the overlay network can tune itself to improve the performance of the scheduling algorithms, according to the incoming workload characteristics and the static attributes of the nodes (number of cores, CPU frequency, amount of memory and storage, etc.). All in all, the collective intelligence of the overlay network (i) evenly distributes the number of data center and desktop/laptop nodes in the overlay network to increase the system resiliency (assuming that the online presence for the desktop/laptop nodes cannot be guaranteed), (ii) shapes the overlay network in a way that the neighborhood of handheld devices is constituted

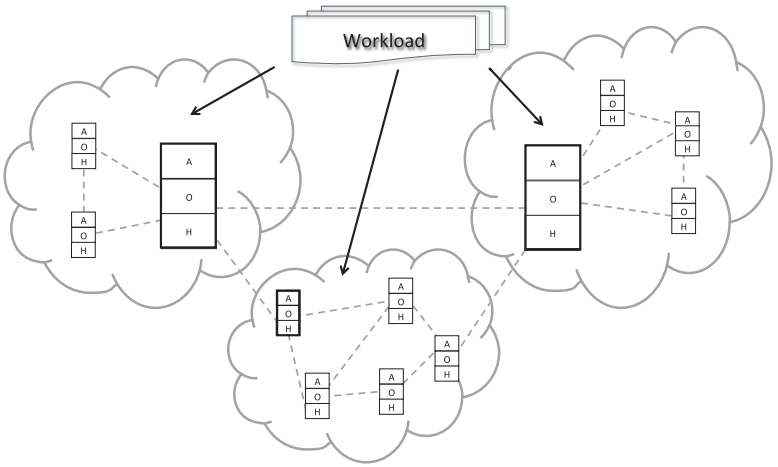


Fig. 2. Volunteer cloud network.

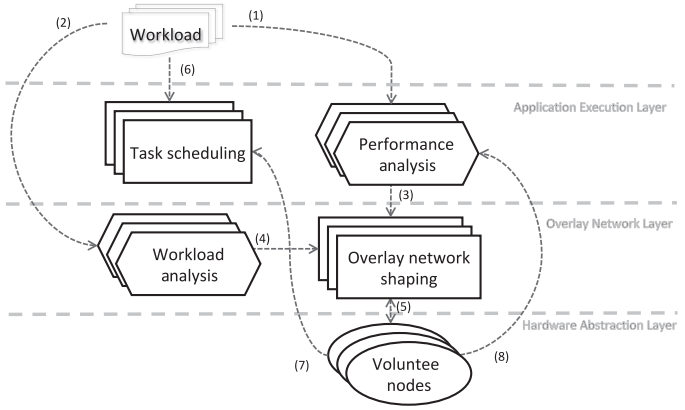


Fig. 3. Collective workflow of agent and layer interactions.

by physically close nodes (with the purpose of reducing the latency, particularly important in the case of virtual/augmented reality applications), or (iii) obtains information from the uppermost layer and performs workload-based partitioning to avoid tasks interference in the execution queues of the nodes. In the presented instantiation of our approach, the overlay shaping technique in Section 3.2 adopts the latter approach (iii), which is then exploited to build a distributed data structure (presented in Section 3.1) that improves the performance of distributed task executions.

Figure 2 shows an example of a volunteer cloud network, where supernodes are highlighted with thick boxes, dashed lines represent the P2P connections, and cloud shapes represent the partitioning of the network into sites.

Our holistic approach is based on the collaboration of nodes according to the workflow depicted in Figure 3, where we distinguish the agents at the application execution and overlay network layers according to their role of *analyzers* (hexagons) or *optimizers* (square boxes). In MAPE-K terminology, analyzers are in charge of the *measure & analyze* activities, whereas optimizers are in charge of the *plan & enact* activities. The information about the incoming workload and volunteer nodes feed several modules. At the application execution layer, the QoS perceived by the tasks (1)

and information from volunteer nodes (8) is considered according to different metrics, whereas at the overlay network layer, the workload information (2) is analyzed for being classified according to its characteristics (e.g., frequency of the task execution requests, required parallelism, SLA). The results of such analysis (3, 4) and incoming information (5) feed the agents' knowledge, thus allowing them to optimize the structure of the overlay network. At the same time, task scheduling is optimized considered the perceived QoS (6) and the hardware characteristics of the volunteer nodes (7). Such a workflow is executed online and is kept active throughout the entire life of the volunteer network.

Workload and performance analysis are realized through the nodes' collective knowledge. As a matter of fact, it takes place by enriching the periodic message exchange between nodes and the corresponding P2P supernodes (used to keep the overlay network alive) with information concerning the performance and task traffic perceived locally by each node. P2P supernodes collect, classify, and analyze information about the workload generated/executed by each node, and the overall system performance. The runtime evaluation of system performance is a straightforward process (once having collected the performance perceived by each node). The online workload analysis and classification is outside the scope of this work, but we refer the interested reader, for example, to Dong et al. [15] and Moro et al. [35].

Several analyzers and optimizers at the different layers can be combined. For example, at the application execution layer, we can use the ACO-based approach presented in the following (see Section 3.1) or any other task scheduling algorithm, such as a random one or a round-robin one. At the overlay network layer, instead of the proposed workload-based partitioning (see Section 3.2), we can choose a P2P structure according to the nodes' geographical proximity [18] or any other well-known P2P protocol, such as Kademlia [23], JXTA [21], or Chord [53]. As a matter of fact, some of the algorithms used in the comparison of Section 4 are a combination of a scheduler with a randomly arranged P2P network.

Thanks to its high degree of flexibility, our holistic approach can be used even to enhance the performance of the standard P2P protocols mentioned previously. In the case of workload-based partitioning, the only requirement to seamlessly adopt our approach is that the P2P network will be arranged in a semihierarchical manner. In this way, the supernodes can easily perform the analysis and negotiate the required modification to the overlay network, to optimize system performance.

3 OPTIMIZERS FOR APPLICATION EXECUTION AND OVERLAY LAYERS

In this section, we present two optimization algorithms adopted, respectively, at the application execution and overlay layers. We describe the application execution layer optimization (originally presented in Sebastio et al. [46]) in Section 3.1 and the overlay layer optimization (originally presented in Sebastio and Scala [49]) in Section 3.2. Then we describe the key aspects of their novel combination in Section 3.3.

3.1 ACO-Based Task Scheduling

The main idea of the approach we presented in Sebastio et al. [46] is to maintain and exploit a distributed data structure that we call the *colored computational field*, which works as a sort of distributed routing table where links are annotated with rates related to the likelihood to find computational resources by following them. The idea is inspired by ACO techniques, where "ant" agents release pheromone tracks when resources are found. Such a pheromone is used by all other ants to easily gather the resources. Our approach uses *colored scout ants* to discover computational resources in the volunteer network and maintain the computational field. Scout ants are continuously spawned to adapt to the variability of the volunteer cloud. It is worthwhile to remark that several algorithms can be defined to exploit the field. Here we focus on a suggestive

```
∀ color k ∈ 𝒦 with period_k: ant_k.coloredAntStep(sourceNode);                    1
                                                                                 2
when n ∈ N generates a task t:                                                   3
  if (n cannot execute t):                                                       4
    while (an executing node n′ is not found) && (∃ hunterAnt attempts):          5
      ant.AntStep(n′);                                                           6
```

Listing 1. Ant main cycle

one where *hunter ants* help to accommodate task execution requests by exploiting the field. Such ant agents navigate the network in search of a volunteer node that can serve a task execution request by choosing their next hop with a probabilistic selection, weighted according to the level of pheromone of the links in the field. Such an operation, called *stigmergy*, may eventually lead to optimality in static networks but may also suffer (as all ACO-based approaches) from stagnation [52], especially in dynamic networks. Stagnation occurs when the ants converge to an apparently optimal decision, which may prevent the system from adapting to the emergence of new and better solutions. Our ACO-based algorithm features some standard techniques to prevent stagnation, such as *evaporation* (pheromones are regularly decreased), as well as some novel ones, such as *temperature regulation* (the likelihood of exploring new paths is increased when the network is updated), *memory aging* (in analogy with the standard aging, releasing pheromone quantities in inverse proportion to the distance to resources), and *angry ants* (a third kind of agent that removes pheromones along outdated links).

Colored scout ants (Listing 1, line 1) periodically explore the neighborhood of a node to discover computational resources and to update the field accordingly. Such ants are specialized by computational pheromones (set K): each color k corresponds to one of them. Instead, hunter ants (Listing 1, lines 3 through 6) are spawned when a task execution request is issued. They exploit the pheromone field to find a volunteer node and update the field according to the received feedback.

3.1.1 Colored Scout Ants. Colored scout ants are periodically spawned in a process that is independent from the request and execution of tasks (Listing 1, line 1). Their goal is to explore the network and update the pheromone field. Each ant releases and follows its own pheromone color ($k \in K$). Listing 2 describes the behavior of scout ants by means of pseudocode. Each scout

```
ant_k.coloredAntStep(Node n){                                                    1
  ant_k.pathAdd(n);                                                              2
  ant_k.saveNodeGoodness(n, k);                                                  3
  ant_k.updateTtl();                                                             4
                                                                                 5
  if (ant_k.getTtl()>0)                                                          6
    nextNode := choose with probability p_k((n, nextNode), E_n \ {ant_k.path()}); 7
    if (nextNode != null)                                                        8
      ant_k.coloredAntStep(nextNode);                                            9
      return;                                                                    10
  previousNode := ant_k.getPreviousNode();                                       11
  ant_k.coloredAntStepBack(previousNode, n);                                     12
}                                                                                13
                                                                                 14
ant_k.coloredAntStepBack(Node to, Node from){                                    15
  new_φ_k := ant_k.getMemoryAgingPheromone(to, from);                            16
  if ( (new_φ_k > φ_k(to, from)) || (k != FINISHING_TIME ))                      17
       φ_k(to, from) := new_φ_k;                                                 18
  previousNode := ant_k.getPreviousNode();                                       19
  ant_k.coloredAntStepBack(previousNode, to);                                    20
}                                                                                21
                                                                                 22
ant_k.getMemoryAgingPheromone(Node to, Node from){                               23
  memoryTrace := φ̃(subPath(to, from));                                           24
  best_φ_k := max(memoryTrace);                                                  25
  return agingDiscount(pathLength(best_φ_k, to));                                26
}                                                                                27
```

Listing 2. Colored scout ant algorithm

ant explores the network (line 7), probing the neighborhood goodness while going away from its home node (the one that spawned the scout ant). Each ant has an associated time to live (TTL), which establishes the number of hops an ant must try to perform during its exploration, before returning home. The TTL prevents endless and unnecessary exploration efforts. When its TTL is exhausted, the scout ant returns back to its source node (line 12), releasing the pheromone according to a memory aging approach (line 23). In the following, the main features of the algorithm are explained in detail.

Choosing the next hop: Temperature-dependent exploration and exploitation. Ant behavior is based on online reinforcement learning (RL) [41], where at each step the decision of which link to explore next involves a choice between *exploration* (trying to gather new information) and *exploitation* (focusing on the best decision, according to current information). Exploration may be considered as a risk run by the node, with the hope to obtain better knowledge and thus make better decisions in the future. A common approach to face the "exploration-exploitation dilemma" is the use of a *softmax* method [41]. Each ant selects its next hop by taking into account both the past path desirability (exploitation) and the exploration compliance, considering the rate $\pi_k(e)$ assigned to each outgoing edge e according to the following equation:

$$\pi_k(e) = \exp\left(\frac{\Phi_k(e)}{T_i}\right), \tag{1}$$

where $\Phi_k(e)$ is the amount of pheromone of type k in the edge e and T_i is a temperature factor described later. Given the rates, the probability $p_k(e_j, E')$ that the k-colored scout ant at node i chooses the outgoing edge e_j is then defined by

$$p_k(e_j, E') = \frac{\pi_k(e_j)}{\sum_{e_q \epsilon E'} \pi_k(e_q)}. \tag{2}$$

Typically, E' is selected to contain all outgoing edges at node i, excluding the one from which the ant arrived (see Listing 2, line 7).

According to the softmax action selection method, we have chosen the Boltzmann/Gibbs distribution, with a tunable temperature function T_i, to probabilistically choose the next hop from node i, while taking into account the expected reward (i.e., the probability to find a node willing to perform a task). The temperature function controls the exploitation/exploration trade-of—for instance, if $T_i \to \infty$, the ant at node i tends to follow a more random approach (all paths have the same preference), whereas if $T_i \to 0$, the ant follows a greedy approach, which reduces the exploration component. Instead, if T is close to 1, the choice tends to be proportional to the pheromone of each link.

One of the roles of the temperature is to prevent stagnation. Indeed, if we choose the temperature to be a monotonically decreasing function with respect to time, then as time goes by, it is possible to reduce exploration and make a more sound use of the knowledge gathered so far. However, every time a new neighbor connects to a node i, the corresponding function T_i should be reinitialized to encourage the exploration of new resources.

Discounting the distance: Memory aging. Scout ants explore the network and record the *node goodness* (or *nest* value, i.e., the resource value associated to the corresponding color) found during their exploration. While returning home, a scout releases a pheromone value, depending on its memory aging factor (to prevent stagnation) and the node goodness in that part of the network (Listing 2, lines 23 through 27). We do not use the traditional concept of aging, where ants deposit less and less pheromone as they move from node to node, because in our setting, the

information that the pheromone provides is useful not only for the node from which the scout has been spawned but also for scout ants spawned by other nodes.

However, to take into account the distance between a potential task execution requester and the node holding the necessary resource, our memory aging mechanism releases an amount of pheromone that is inversely proportional to the distance from the best resource found so far rather than to the distance from the node that spawned the ant (as in traditional aging). In other words, our memory aging mechanism considers what the ant remembers from the goodness of the best node, in the subsequent portions of the path it has followed. This can be achieved, for instance, by implementing the function `agingDiscount(mem_aging on best_`ϕ) (see Listing 2, line 26), as `best_`ϕ − `mem_aging` · `AgingFactor`, where `best_`ϕ is the best value found so far, `mem_aging` is the distance from it, and `AgingFactor` is the discounting factor.

Discounting information age: Evaporation. In addition to the dynamic temperature and memory aging mechanisms, we also use the evaporation technique to deal with stagnation in the presence of volatile resources.

The finishing time, for instance, is a volatile resource measure, and its value should be updated frequently. A higher amount of pheromone is assigned the more the declared finishing time is closer to the current time. Further, pheromone is released only if the new value is higher than the previously released one. Instead, if pheromone values would be updated regardless of the best previously found values, they would be highly variable and thus would cause unstable information.

We consider resources such as the amount of RAM and CPU to be nonvolatile, as they cannot be allocated forever but only on a short-term basis (i.e., to execute tasks). Thus, until the node participates in the network, its resources are stable, and the corresponding deposited pheromone does not need to be updated by means of evaporation. When a node perceives a new neighbor, the former increases its temperature to update the field. Instead, when a node notices that one of its neighbors has left, it uses angry ants (described in the following) to update the field.

Dealing with dynamic networks: Angry ants. Despite the nonvolatile nature of resources, the unstable nature of the network of participants [54] can lead to stagnation: when a node that caused the update of the pheromone on several links goes offline, all subsequent task execution requests on the nodes connected with those links may follow a wrong path, without finding the desired resources. As a remedy, we propose *angry ants*, which are spawned by scout ants when they find an abrupt change in the field. Angry ants follow back the path of colored scout ants, and throw away a certain amount of pheromone of the corresponding color, to force the update of the corresponding pheromone color by future scout ants.

3.1.2 Hunter Ants. When a node has a task for which it cannot respect the deadline, it starts spawning multiple *hunter ants* (Listing 1, lines 3 through 6). Every hunter ant tries to find a node ready to satisfy the task execution request, by exploring the network according to the field and the task characteristics. Task execution requests are sent to nodes that have been found by the hunter ants, until one of them accepts or the hunter ant attempts are exhausted. The hunter ant brings with it only a task description with its functional and nonfunctional requirements, not the task itself, to minimize transmission overheads.

The behavior of hunter ants is sketched in Listing 3. Each hunter ant tries to find a node willing to execute the task (line 4), by following the colored computational field (line 8) built according to the overall pheromone (see the explanation of Equation (5), shown later). If the hunter ant does not find any node willing to collaborate after its TTL, it returns to its home node. In the following, we provide a detailed explanation of the main features of the algorithm.

```
ant.antStep(Node n, Task t){
  ant.pathAdd(n);
  ant.updateTtl();
  if (askExecutionToNode(n, t))
    previousNode := ant.previousNode();
    ant.antStepBack(previousNode, n);
  else if (ant.getTtl()>0)
    nextNode := antChooseContact(p_k(E_n \ {ant.path()}), t);
    if (nextNode != null)
      ant.antStep(nextNode, t);
      return;
   ant.antStepBackHome(ant);
}

ant.antStepBack(Node to, Node from){
  ant.depositPheromone(to, from);
  previousNode := ant.getPreviousNode();
  ant.antStepBack(previousNode, to);
}

depositPheromone(Ant ant, Node from){
  new_φ_fb := ant.agingPheromone(to);
  φ_fb(e_from) = new_φ_fb;
}
```

Listing 3. Hunter ant algorithm

Combining and minimizing resources: Resource allocation heuristic. Choosing the next hop, hunter ants take into account the pheromones of all colors deposited in the field, which may offer contradictory information. For example, memory-colored and CPU-colored pheromones can promote different links. Our approach is based on a weighted sum of both fields.

An additional issue is that, in an execution-oriented environment as turns out to be the volunteer cloud, the main purpose is to maximize the overall number of tasks that meet their deadline. Such a problem is clearly intractable in a global manner (i.e., even the problem of finding the best task-node match is well known to be NP-complete [59]) and would require perfect predictions of future task arrival times and characteristics, which is totally unrealistic in the complex and dynamic environment of the volunteer cloud, where tasks requests and nodes participating in the network change over time. Therefore, hunter ants use local heuristics, based on the idea that minimizing wasted resources (the ones that are reserved but not completely used) will increase the probability to accommodate more requests in the future. These heuristics rely on two functions: the *single resource waste ratio* (*srwr*) and the *combined resource waste ratio* (*crwr*), defined in Equations (3) and (4), respectively:

$$srwr(x_k, y_k) = \frac{\min(x_k, y_k)}{\max(x_k, y_k)}, \tag{3}$$

$$crwr(\mathbf{x}, \mathbf{y}) = \sum_{k \in 1..|\mathbf{x}|} \frac{\eta_k srwr(x_k, y_k)}{\sum_{\sigma \in 1..|\mathbf{x}|} \eta_\sigma}, \tag{4}$$

where $\mathbf{x}$ and $\mathbf{y}$ represent, respectively, the task requirements and the node resources, whereas the weight vector $\boldsymbol{\eta}$ of size $|\mathbf{x}|$ expresses the importance of each resource. Smaller values of *crwr* suggest a higher mismatch degree between requested and provided resources.

Weighting links. Such heuristic functions are used to associate goodness values to links. The task-t's resource requirements t_Q are expressed as a pheromones vector $\Phi(t_Q)$ applying, for each of the requested resources, the same functions used by the scout ants. Then the goodness of a link e will be based on the value of $crwr(\Phi_{\mathcal{R}}(e), \Phi(t_Q))$, where $\Phi_{\mathcal{R}}(e)$ is the pheromone vector associated to all computational resources in $\mathcal{R}$ (which coincide with those expressed in the QoS of the task). Task requirements that are closer to the available ones are preferable. For a single color, the optimal value is approached when the single resource waste ratio tends to 1, whereas the worst

case is when resources are reserved but not completely used by the task, and the function tends to 0. In the other cases, for each single resource component k, we obtain $\Phi_k(t_Q)/\Phi_k(e)$ when the resource is underused or $\Phi_k(e)/\Phi_k(t_Q)$ when the resource is overused.

Pheromone release. When a hunter ant finds a node willing to perform a task, it releases its own type of pheromone, which serves to record a measure of the node's availability to execute remote tasks, its presence in the network, and also its load. The node's willingness to perform tasks can be regarded as a reputation assigned to the node, and it is subject to pheromone aging and evaporation, to take into account the loss of knowledge about the node behavior. At each hop, a hunter ant computes an overall pheromone value $\Psi(e,t)$ for a candidate edge e according to

$$\Psi(e,t) = crwr^{\alpha}(\Phi_{\mathcal{R}}(e), \Phi(t_Q)) \cdot \Phi_{ft}^{\beta}(e) \cdot \Phi_{fb}^{\gamma}(e) \cdot \lambda^{\delta}(e, t_Q), \tag{5}$$

where Φ_{ft} is the pheromone value associated to the node's finishing time, Φ_{fb} is the feedback pheromone released by hunter ants, and $\lambda(e, t_Q) \in \mathbb{R}^+$ is a heuristic measure that evaluates the estimated performance of link e for a task with QoS t_Q, in terms of data rate and delay perceived in the last interaction along e. The last measure takes into account the network overhead for transferring the task to the node that will execute it. Parameters α, β, γ, and δ are tunable weights for the components of the equation, which are normalized in the range $[0, 1]$.

Exploration. Unlike the function used by the colored ants, hunter ants combine all types of pheromone colors (Listing 3, line 8). However, the probability to choose link e' as the next hop is computed in a similar manner based on the following rate:

$$\pi_h(e', t) = \exp\left(\frac{\Psi(e', t_Q)}{T_i}\right). \tag{6}$$

All in all, our ACO-based approach has been designed with the peculiarities of volunteer clouds in mind: the colored scout ants explore the network to build distributed knowledge (something that would be irrelevant in the case of standard cloud computing systems with a few well-known data centers); the memory aging mechanism works considering the best resources found in the ant's path (and not according to the distance from the node that spawned the ant as in the traditional aging); the evaporation mechanism is useful for dealing with volatile resources (since the knowledge on the execution queue is highly variable and changes every time a new task is accepted for execution and when a task in queue completes its execution); the angry ants clear a portion of the distributed knowledge to take into account the dynamic online presence of the nodes (again something not really relevant in standard cloud systems).

3.2 Dynamic Workload-Based Partitioning of the Overlay Network

Task distribution strategies applied to the cloud or volunteer computing environment assume the presence of a central *load balancing node*, which knows the available resources and their current load. Exploiting such global knowledge makes it possible to optimize task execution by allocating each task to the node that can complete it first. Unfortunately, maintaining such a global knowledge is infeasible in volunteer clouds due to the large size of such systems and the high dynamism and heterogeneity of the resources. Even for large cloud providers, whose data centers are constituted by many nodes placed at different geographical locations, maintaining an overall knowledge is a hard job. To relieve the knowledge requirements, a widely adopted approach foresees the use of *cloud partitioning* [27, 64]. The simplest and most spread approach is partitioning the cloud according to geographical location. In this way, the knowledge can be managed at two levels. At a higher level, the load balancer knows only the overall load and capabilities available in each cloud site. At a lower level, each site manager is in charge of maintaining accurate knowledge

on the status of each node that constitutes the site. When an execution request arrives in a cloud site, the cloud manager evaluates the site load. If the site can satisfy it, the request is executed locally; otherwise, it is forwarded to the central load balancer. The load balancer builds a priority queue with all cloud sites ordered according to their actual load, then forwards the request to the less loaded site. Within each site, the same approach can be replicated on a smaller scale or more sophisticated distribution protocols can be adopted.

This partitioning approach works well when the number of nodes is known a priori. However, in a volunteer cloud system, the number of participating nodes can be higher than in a classical cloud. Moreover, resources are heterogeneous, and their online presence cannot be guaranteed for the whole duration of the task execution. Moreover, due to the volunteer participation of each node, the geographical distribution of resources may not be uniform.

Fortunately, the task workload usually exhibits patterns that can be classified. In Sebastio and Scala [49], we exploited such patterns to obtain a simple dynamical partitioning of the volunteer cloud in sites according to the task characteristics. It is worthwhile to observe that, differently from the central load balancing approach used in classical cloud partitioning, here an accurate knowledge on node characteristics or load is not required.

Differently from static geographical partitioning, with this strategy the volunteer cloud is dynamically partitioned in logical sites (disregarding from the physical location of the machines) according to the incoming workload (e.g., to avoid task interference in the execution queue without requiring a priority queue). Indeed, assuming the presence of two types of tasks (as it has been proved in the Google workload [32]), interferences in the execution queue of long running tasks (which are less restrictive on QoS requirements) with small running tasks (with restrictive QoS requirements) could be generated. In fact, in our previous works [3, 46], we analyzed different task distribution protocols and observed a saturation in the requests that were satisfied. In these scenarios, even increasing the number of executing nodes, the volunteer cloud performance (evaluated as the number of executed tasks that respect their deadline) does not improve.

The P2P overlay network of the volunteer cloud is then divided and assigned to a supernode that becomes responsible for a given site. In addition to the typical duties of a supernode (e.g., registering the nodes that enter and leave the network, mediating the communication between sites), each supernode negotiates with the other supernodes to evaluate the benefit of a site resizing. For example, if a site perceives underutilization, easily satisfying all requests for the task type assigned to it, the supernode in charge of that site can evaluate the migration (in the overlay network) of a certain amount of volunteers to another site that instead is overloaded and unable to satisfy all of the requests. Moreover, supernodes can interact to tune the right number and size of each site according to any other performance metric. A deeper discussion and a pictorial representation of these interactions is provided in Section 3.2.1 and in Figure 4.

It is worth noting that the supernode does not need to know the actual load of each node in its cloud site, but only to have an overview of the site performance and the workload characteristics. For example, it is possible to design a mechanism where an underloaded or overloaded node, after a given time period, signals its status to the supernode. If the supernode receives a certain amount of these requests, in a given time frame, it can interact with other supernodes to resize the sites. In our previous work, we considered nodes that accept or refuse remote execution requests according to the perceived task's miss rate [3]. The same approach is adopted here to trigger the dissemination of load information to the supernode.

Other approaches to avoid interference of task types can act locally, on each volunteer node, adopting a specific queueing policy (e.g., priority queueing). Unfortunately, the possible benefits of a priority queue are paid in terms of increased computational costs, as a queue check/update is

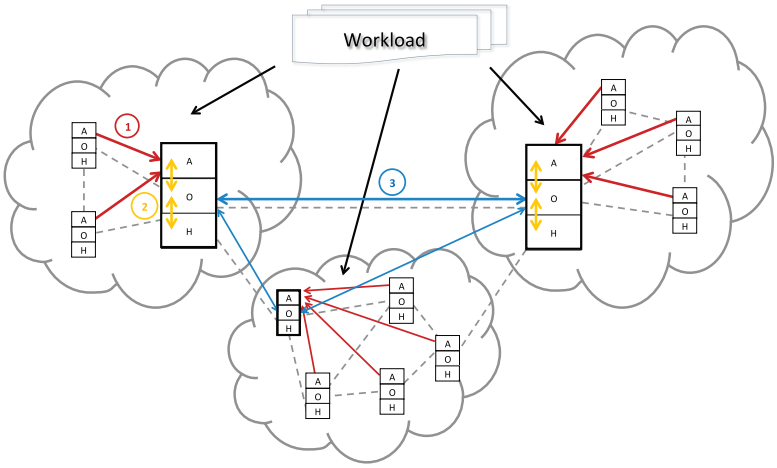


Fig. 4. Agent interactions to partition the overlay network.

required every time a new task arrives in the system or an execution is completed, thus making their application infeasible in such a volunteer cloud context.

3.2.1 A Heuristic Approach to Tune the Cloud Sites. Here we exploit a heuristic that resizes the partitions by evaluating the perceived performance with respect to the metrics of interest. The pseudocode of such a heuristic is reported in Listing 4.

```
Data:
M: performance metrics = m' (task-type dependent) ∪ m'' (overall metrics)
i: index on metrics. Metric indexes are ordered according to their importance (i=1 is the highest)
j: index on cloud sites/task types

cloudSitesTuning(){  //multi-objective approximate optimization
  i = 1;  //init
  while(TRUE){
    wait(EVAL_PERIOD);
    lastPerformance = evalPerformance();

    //order the sites according to their importance for the current metric:
    if (metric_i in m''){ //if the current metric of interest does not regard a single task-type
      evalWorkload();
      order_j_according_to_the_evaluated_workload //the importance of each site is then given from their
            participation to the overall workload and not chosen by metric_i
    } else //metric_i in m'
      order_j_popping_the_site_corresponding_to_i_on_TOP //metric_i explicitely refers to a task-type

    resizeCloudSite(feedback)
    wait(EVAL_PERIOD)

    if (lastPerformance[metric_i] == evalPerformance()[metric_i]) //eval the results of the last tuning
      i++ (cycle on i) //try to improve the next metric since the current one has not been improved
    else
      feedback = (lastPerformance[metric_i] < evalPerformance()[metric_i]) ? POSITIVE : NEGATIVE;
  }
}

evalPerformance(){
  sum_{i}^{M} w_i * metric_i; // performance metrics are weighted
}

resizeCloudSite(feedback){
  ∀ first j: cloud_site_j = %INC*feedback;
  ⟹ ∀ last j: cloud_site_j = %DEC*feedback;
}
```

Listing 4. Cloud Sites Tuning Heuristic

The heuristic works through a two-level decision-making process. At the higher level, supernodes collaborate to characterize the overall workload (*online workload classification*) and to tune the cloud (in terms of size and number of partitions). At the lower level, in each cloud site, the supernode pinpoints the cumulative load of the partition and the corresponding perceived performance. The number of cloud partitions is determined according to the task types identified by the online workload classification but, in general, are chosen by the supernodes.

The M measurable performance metrics are constituted by a subset m' that concerns a single task type (e.g., the waiting time per short tasks), and by m'' that instead does not regard any specific task type (e.g., the overall hit rate). The volunteer cloud manager assigns weights to each performance metric (line 29) defining a (multi)objective optimization function. According to the assigned weights, the tuning function (line 6) tries to improve the performance of each metric, providing more nodes to the site that results to be more critical for such a metric. If the metric belongs to m', the corresponding site is increased, whereas if the metric belongs to m'', the workload characteristics are evaluated to understand which are the most important task types to improve the corresponding metric. For example, if the metric is related to the overall hit rate and the workload is mainly constituted by small tasks, it could be worth evaluating an increase of the corresponding site. After each resize action (line 33), the cloud performance is re-evaluated (line 22) to obtain positive or negative feedback on the tuning performed on the sites. If the performance has not been changed according to the current metric, the subsequent metric of interest is considered in the subsequent optimization cycle.

3.3 Combining ACO-Based Task Scheduling and Workload-Based Partitioning

For illustration purposes, in this section we describe how the best algorithms that we have investigated in our previous research efforts (i.e., the two techniques presented in the previous sections) can be combined in an effective way. The overlay network optimizer structures the overlay network in such a way that nodes are grouped according to the type of tasks to which they are more suited. This solution eases the activity of hunter ants, as they can directly jump to the most promising region of the network (i.e., the one populated by nodes specialized in the type of task to be executed). In detail, in case the hunter ant is generated by a node in cloud partition a, but it must find a node specialized in type b tasks, the hunter ant is moved to cloud partition b without consuming its TTL.

Figures 4 and 5 summarize the key interactions performed by agents to carry out the holistic optimization. They are split in two figures only for the sake of clarity but actually are performed in parallel. Table 1 describes these interactions decomposing them in steps. In the figures, dashed lines represent node neighborhoods, clouds are logical sites/partitions, and nodes with thick borders are supernodes, whereas the workload (consisting of an independent set of tasks) is generated by any set of nodes participating to the network.

Periodically (see step 1 in Figure 4 and Table 1), each node sends to the supernode responsible of its site: (1.A) information about the perceived performance (e.g., hit rate, tasks waiting and sojourn times); (1.B) a description (if any) of the task execution requests that it has generated (composed by number of tasks, SLA, required number of cores, etc.); (1.C) a heartbeat to support the basic P2P functionalities. Each supernode can thus (2.A) aggregate the perceived performance and execute the *performance analysis*; (2.B) perform the *workload analysis* by building a collective knowledge on the incoming workload (i.e., classes of tasks with their characteristics, as later exemplified in Section 4.2.2 and Figure 6). Finally (see step 3), the supernodes, sharing the knowledge on their partition, perform the workload-based partitioning as discussed in Section 3.2.

Instead, at the application execution layer, the colored scout ants (i.e., mobile agents) are spawned by each node (see step 0 in Figure 5 and the second half of Table 1) to collectively build

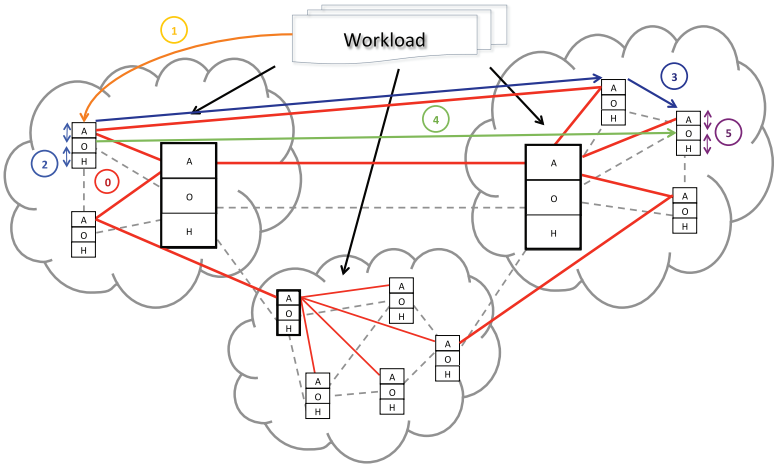


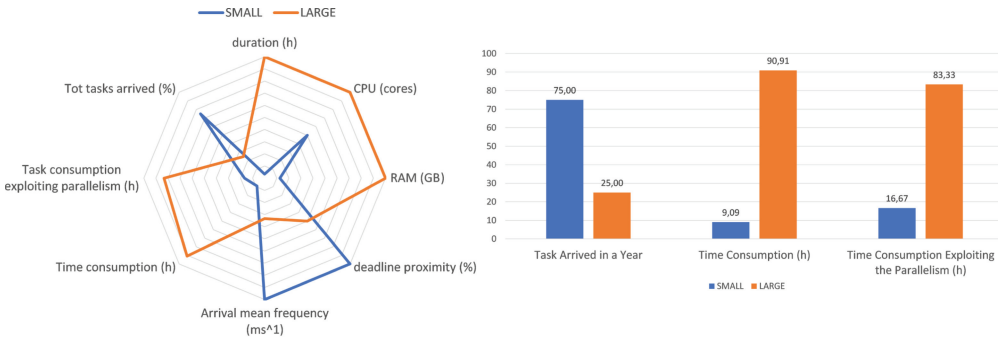
Fig. 5. Agent interactions to schedule the tasks.

Table 1. Agent Interactions in the Proposed Combination

Step #	Who	What	When
1.A	Nodes to supernode	Send the perceived performance	Periodically
1.B		Send tasks descriptions	
1.C		Basic P2P functionalities	
2.A	Each supernode	Aggregates the perceived performance (ref 1.A) (performance analysis)	Periodically
2.B		Builds a collective knowledge (ref 1.B) (workload analysis)	
3	Supernodes	Share the knowledge on their site/partition (workload-based partitioning)	Periodically
0	Each node	Sends colored scout ants to build the colored computational fields	Periodically
1	Any node	A task arrives in the system	On demand
2	Request generator	Check the satisfiability of the request	On demand
3	Request generator	Sends the hunter ants	On demand
4	Request generator to node that accepts	Task transmission	On demand
5	Node that accepts	Task execution	On demand

Note: The first part summarizes the interactions in the overlay network, whereas the second part summarizes the ones in the application execution layer.

the colored computational fields. Scout ants explore the whole network, but as mentioned earlier, thanks to the optimization in the lower layer, the computational field that they build is already optimized to be subsequently exploited by hunter ants efficiently. Every time a new task execution request is generated (step 1), the source node first checks the internal satisfiability of the request (step 2). If it cannot satisfy such a request, it spawns hunter ants (step 3) that, exploring the computational field, quest for a node willing to accept the task execution. Once a hunter ant finds a suitable node, the task is transmitted (step 4) and finally executed (step 5).



(a) (b)

Fig. 6. Workload analysis: attributes comparison for small and large tasks (mean values) (a); percentage of the total amount of tasks arrived in a year, required execution time (in hours), and execution time exploiting the tasks parallelism (b).

4 PERFORMANCE EVALUATION

This section describes the tools used to carry out the performance evaluation, the characteristics of the considered scenario, and the algorithms that have been compared, and concludes by reporting and discussing the obtained results. Resources and information to reproduce our experiments are available in Sebastio [43].

4.1 The Volunteer Cloud Simulator

AVoCloudy [43, 45] is a volunteer cloud simulator, built on top of DEUS [4, 14] and MultiVeStA [50, 63].

DEUS is a flexible open source, general-purpose, Java-based discrete event simulator supporting the analysis of every kind and size of complex system. DEUS is characterized by three basic elements: (i) *nodes* are entities that populate and interact within the system; (ii) *events* define the internal actions of the nodes, as well as the interactions with other nodes and the environment; and (iii) *processes* determine the stochastic or deterministic scheduling of the events.

MultiVeStA is an efficient distributed statistical model checker that can be easily integrated with any existent discrete event simulator. MultiVeStA is equipped with MultiQuaTEx, a flexible language to formally express system properties of interest. MultiQuaTEx queries are evaluated by MultiVeStA by means of independent distributed simulation runs, until the required statistical accuracy is met. MultiVeStA has been chained with several simulators and used to study several different scenarios [63]. The reader interested in a deeper analysis of the performance scaling achievable by integrating MultiVeStA with a discrete event simulator can refer to Pianini et al. [40].

4.1.1 Performance Parameters. AVoCloudy allows one to assess several performance metrics, such as hit rate, total number of execution requests, task waiting and sojourn times, and number of tasks assigned to each node. In the following, we focus on the most significant ones to compare the QoS shown by the considered algorithms: (i) *hit rate*, defined as the relative amount of tasks that met the deadline or that are still running but that will complete successfully if their host will not go offline (i.e., a higher value denotes the ability of a strategy to accommodate a heavier load), and (ii) *mean waiting time*, which is the average amount of time that tasks have spent in the execution queues (i.e., lower values suggest a better system response to incoming requests).

Table 2. Simulation Configuration: Node Attributes

Type	CPU Freq.	Cores	RAM	Nodes
Desktop/laptop	1–2GHz	1–6	0.1–2GB	100–3,000
Data center	1–3GHz	2–32	2–6GB	7

4.2 Simulated Scenario

In the following, we describe the main characteristics of the scenario used in our performance evaluation. We assumed a set of universities participating to the volunteer cloud with their data centers, whereas desktop/laptop resources are shared by laboratories and private users when idle (e.g., at night) or used for Web browsing only.

4.2.1 Node Characteristics. The network of participants initially includes 10 cloud sites, 7 of which have a central data center and several desktop/laptop nodes, whereas the rest of the sites are composed by desktops/laptops only. The specification of node resources is reported in Table 2. Obviously, desktop/laptop nodes are less computationally powerful, as they correspond to private devices.

We consider different cloud configurations, which differ in the number of participating volunteer nodes (from 100 to 3,000), each one belonging to one cloud site. Every site is managed by a supernode, which can be run on top of a data center or a volunteer node. The overlay network is semihierarchical, with supernodes connected with peers of other sites and normal nodes having connections within the same site only. Each node joining the network notifies its presence to the corresponding supernode and receives a list of neighbors (i.e., a random subset of the volunteer nodes in the same site). The only exception to this overlay network configuration takes place with the proposed algorithm, in which the overlay network is partitioned according to the characteristics of the incoming workload (as described in Section 3.2).

Nodes are both task producers and consumers, and share their resources to address task execution requests coming from other nodes but can also create their own requests. Tasks are executed in exclusive application environments. A task is accepted for execution only if its timely completion can be guaranteed, (i.e., if the node that has received the task execution request, considering all tasks already in its execution queue, can satisfy the deadline specified in the SLA); otherwise, the task is discarded. A completed task marks a hit for the node on which it has been executed. The cost of communication is computed by means of the simple yet realistic network models described by Saino et al. [42]. Moreover, we assume, for performance reasons, that tasks are executed only if the node can satisfy the RAM requirements (i.e., we forbid the use of memory swapping techniques that could have a profound impact on the perceived QoS).

4.2.2 Workload Characteristics. We have adopted the Google Cloud Backend workload model [22], described by Mishra et al. [32]. There, task requirements are characterized by CPU cycles and memory occupation. The task attributes that we have considered are reported in Table 3, namely task duration, required number of cores, RAM, deadline offset, and arrival mean. As workload data are partially obfuscated [22], we made some assumptions. An example is the QoS, in terms of task deadlines, after which task executions are considered to be useless. We set up a deadline offset with respect to the actual duration of the task. The mean arrival characterizes the task arrival process, which is Markovian, as derived by Mishra et al. [32] from the Google Cloud Backend traces. The interarrival time between two consecutive tasks is modeled as an exponential random variable with a mean value equal to 600ms for large tasks and 200ms for small tasks. From a queue-theoretic point of view, the scenario can be seen as a queueing model where data centers

Table 3. Simulation Configuration: Task Attributes

Type	Duration (hours)	CPU (cores)	RAM (GB)	Deadline Offset (%)	Mean Arrival (ms)
Small	0–0.4	1	0–0.5	0.2	200
Large	1–12	1–4	1–4	0.4	600

are modeled as M/G/m/1 queues, whereas desktops/laptops are modeled as M/G/1/1 queues. For instance, task arrivals are modeled by a Markovian process (M), service time follows a generic (G) distribution, data centers have m virtual machines (VMs), desktops/laptops have 1 VM each, and task queues are unbounded. The duration of the simulated scenario is 1 hour, with 10ms granularity. In this article, the focus is on building distributed knowledge of the available machines and optimizing their usage according to the incoming workload, in the presence of a large number of nodes (up to 3,000 in our scenario). Thus, we have assumed that for the simulated time window (i.e., 1 hour), the online presence of the nodes could be reasonably considered stable (with just a few nodes that join and leave, corresponding to a very small number of task misses).

In Figure 6, the characteristics of the Google workload are analyzed. In particular, it is possible to observe (see Figure 6(a)) that large tasks are more resource demanding (in terms of CPU cycles, cores, RAM) than small ones but have more relaxed deadlines and are generated less frequently. According to a Poisson arrival distribution, in 1 year only a quarter of the generated tasks are of the large type (see Figure 6(b)). However, weighting task arrival with required CPU cycles gives one a very different view: small tasks demand for only 9% of the total computation. Finally, considering that large tasks are more capable of exploiting CPU cores, time consumption is divided at 17% and 83% for small and large tasks, respectively. This last observation, combined with the knowledge of task deadlines, has brought us to partition the overlay network according to the incoming workload, to avoid that small tasks get stuck by large tasks in the execution queues.

4.3 Scheduling Algorithms

The proposed holistic approach to support execution-oriented tasks has been compared to other standard algorithms, namely random, round-robin, local diffusion, and greedy oracle. With the *random* approach, execution requests are randomly spread to the neighborhood of the node that generated the task, not considering task requirements or node resources. The *round-robin* approach probes the neighborhood in a circular order, whereas *local diffusion* is a variant of our ACO algorithm that replaces scout ants by local diffusion of pheromones, according to a procedure inspired by spatial computing [65].

The *greedy oracle* algorithm is impossible to realize in practice, as it assumes to have complete information (knowing all nodes' resources and queued tasks) and assigns the incoming task to the node that can complete it first. Thus, from local optimal choices, the greedy oracle algorithm tries to obtain a global optimum. We use this algorithm as a sort of upper bound to better evaluate how close the algorithms are to near-optimal performance.

4.3.1 ACO Task Scheduling in a Workload-Based Partitioned Network. As described in Section 3.1, our ACO-based algorithm is highly parametric. The scout ant behavior can be tuned acting on its configuration through the choice of several of its parameters, such as TTL, initial pheromone on each path, pheromone depositing and aging functions, Boltzman's temperature function, and evaporation rate. Some of the configuration parameters of the algorithm are functions (i.e., releasing, aging, and temperature) for which the current implementation considers several possibilities (constants, linear or exponential functions, user-specified functions, etc.). AVoCloudy allows one to easily express ant configurations by means of an XML file.

The configuration of the pheromone deposit function depends on the color. In our performance evaluation, for computational resources (CPU frequency, CPU cores, and memory), the function is defined as x, whereas for color "finishing time," the function must be decreasing (to assign more pheromone when the finishing time is closer to the actual time), and thus it is configured with $1 - x/5$ and with a constant evaporation rate of 0.0001. Scout ants are spawned with a period of 50 seconds. Hunter ants are instead configured with three attempts for each task (hunting efforts before giving up), a pheromone deposit function equal to $1 - x$, a weight for each kind of pheromone (used in Equation (5)) equal to 1, and a constant temperature value of 1.

According to the workload analysis described in the previous section, we partition the overlay network dedicating 70% of the nodes to the sole execution of large tasks, whereas the remaining 30% is devoted to small tasks.

Furthermore, to evaluate the benefit of the proposed holistic approach with respect to a simpler task scheduling algorithm, we compare it to the use of ACO scheduling alone.

4.4 Results

Apart from the basic common configuration that we described previously, it is worth mentioning that every node uses ants that are configured with exactly the same behavior, despite that AVo-Cloudy allows defining groups of peers with different strategies, and we plan to study this case in a future work. We performed parametric simulations to study the behavior of the system for different numbers of participating volunteer nodes (on the horizontal axis). Obviously, the higher the number of nodes, the better the performance of the system in terms of hit rate.

In the following, we refer to the average results obtained after reaching a 95% confidence interval, with a radius of 0.001, evaluated with Student's t-test. To reach the desired confidence interval, MultiVeStA automatically decides how many simulation runs are necessary. In our experiments, about 20 simulation runs (with different seeds) were necessary. Each simulation implying the ACO algorithm took several hours. The presented results should be considered only for the sake of example, as a more careful parameter tuning should allow to obtain even better performance results. In particular, the workload-based approach to partition the overlay network can consider different optimization criteria (and even a combination of them). Being the workload dominated by small tasks, to achieve a better overall hit rate, the algorithm has favored such type of tasks and then slightly penalized the large ones.

When the number of nodes is higher, more scout ant explorations are required to build an informative computational field and make better decisions. The basic local diffusion algorithm considered does not seem to build a very informative field and consequently underperforms the ACO algorithm. The overall number of performed tasks is acceptable, considering the limited number of participants—that is, even if in some cases the hit rate is not close to 1, in most cases it is close to the hit rate of the fictitious greedy oracle algorithm. Moreover, one should take into account that, despite that it is very likely that the Google cluster is able to satisfy the entire workload, the number of machines in the largest volunteer network considered in our performance evaluation is significantly smaller than the number of machines available in the Google cluster to which the workload model belongs (that are in the order of 10,000).

We observe that in the overall hit rate (Figure 7, bottom) among the scheduling algorithms, the best performance is achieved by the ACO algorithm, but our holistic approach combining ACO and overlay partitioning outperforms it even in the presence of a limited number of volunteer nodes. Such a benefit is mainly achieved because of the improvement in the hit rate of small tasks (Figure 7, top left), whereas the hit rate of large tasks is essentially the same (Figure 7, top right). Once more, we recall that the workload is mainly composed of small tasks.

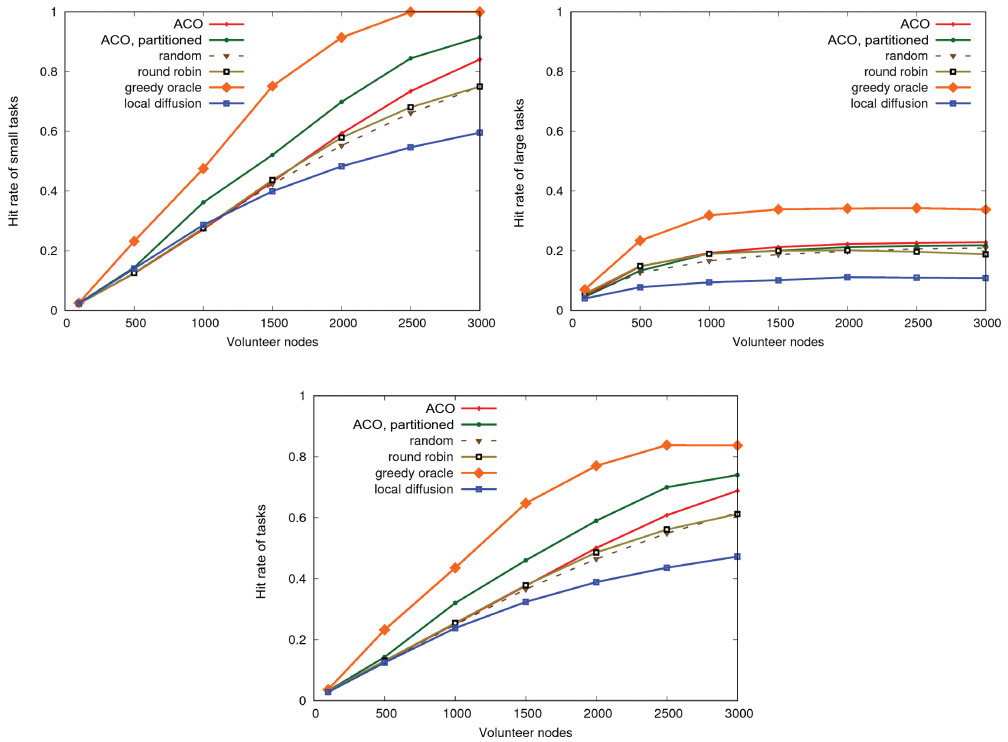


Fig. 7. Hit rate for small (top left), large (top right), and all (bottom) tasks.

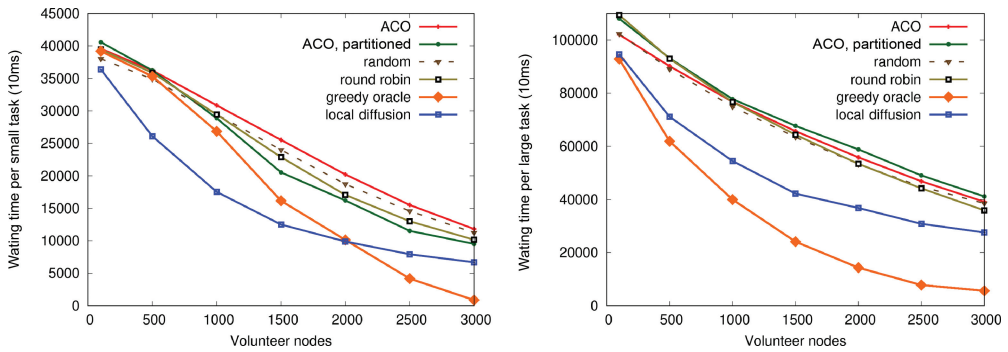


Fig. 8. Mean waiting time for small (left) and large (right) tasks.

The performance gain achieved by our approach is explained by observing the task waiting time (Figure 8). Having removed the interference among task types, in which large tasks (with high computational demand but lower deadline requirements) obstruct the execution of further small tasks, the small tasks are able to perceive a significant improvement in the waiting time (Figure 8, left). The waiting time for the large tasks is only slightly affected (Figure 8, right).

For the sake of brevity, we do not show the results related to the other performance indexes. Nevertheless, we summarize the insights we obtained. With a reduced number of nodes, the knowledge added by the scout ants and the mismatch policy followed in Equation (4) tend to favor large

tasks to data center nodes, leading to increased waiting and sojourn times, with a lower execution rate, for small tasks. Thus, large tasks become a bottleneck for small ones. Scout ants provide an almost linear scaling of executed tasks by increasing the number of nodes and the number of accepted remote requests. With them, the load is better spread among the nodes that are able to execute the tasks.

All in all, our ACO algorithm offers the best performance in terms of perceived QoS at a reasonable price in terms of communication overhead, but it is outperformed by the proposed holistic approach thanks to the optimization in the overlay network.

Finally, it is worth noting that the Google workload is compute intensive rather than data intensive. Indeed, in the latter case, the task transferring process could have been more affected by latency, bandwidth, or lack of geographical locality in the design of the network overlay partitions.

5 RELATED WORK

In our recent research efforts, we tackled the problem of collaborative task execution in the volunteer cloud [44] from several perspectives. In Sebastio et al. [48], we proposed a framework to allocate task requests according to different policies defined by suitable integer programming problems. Those problems are solved in a distributed fashion, relying on the alternating direction method of multipliers. An example policy definable in our framework is the maximization of the node greenness [47]. A different point of view is considered in Celestini et al. [11], in which we proposed a methodology to assess, at design time, the performance of collaborator selection strategies in a scenario where cloud participants are willing to share their machines but are not willing to disclose details of their resources.

In the remainder of this section, we briefly discuss some of the related contributions that can be found in the literature, particularly on the use of ACO-based optimization, partitioning techniques in cloud computing, and cloud simulators.

5.1 ACO-Based Approaches

The ACO approach was first proposed by Di Caro and Dorigo [9] to address the routing problem. In their AntNet algorithm, each artificial ant builds a path from source to destination. While building the path, ants collect information about the time length of the path components and implicit information about the load status of the network. Although our ACO algorithm is clearly inspired by that work, it does address the more complex problem of distributed QoS-constrained task execution.

In their comprehensive survey on approaches to network routing and load balancing based on ACO, Sim and Sun [52] stressed the main weakness of ACO-based approaches, namely *stagnation*, and focused on the many strategies that have been developed to deal with it. In addition to the ones featured also by our algorithm (namely evaporation and aging), they considered *pheromone smoothing* (placing a maximum to the amount of pheromone and releasing less pheromone when that threshold is closer), *pheromone limiting* (setting upper bounds on the amount of deposited pheromone), *privileged pheromone laying* (in which a privileged set of ants may release more pheromone than the rest), and *pheromone-heuristic control* (in which the choice of ants is a weighted combination of the amount of pheromone and the estimate of a heuristic).

In the following, we describe some of the efforts on solving load balancing problems in task distribution systems through the use of ACO-based approaches.

Mishra and Jaiswal [33] proposed a simple ACO approach to deal with the load balancing problem, to have every node doing approximately the same amount of work at any instant of time. The proposed ACO-based algorithm for dynamic load balancing relies only on the current state of the

system (no prior knowledge is needed). Each node is configured with its capacity, its probability of being a destination, and its pheromone (or probabilistic routing) table, whose role is similar to the one of our colored computational field. Each row of the pheromone table is a routing preference for each destination, and each column represents the probability of choosing a neighbor as the next hop. Ants are launched with random destination, to feed the information of the table. When an ant reaches a node whose pheromone table is empty, it does make a random decision. An extended version of such an algorithm considers the presence of multiple ant colonies, with the sole purpose of reducing the likelihood that all mobile agents establish the same connection. In our opinion, although such an approach is suitable for load balancing in network routing problems, it is not adequate for collaborative task execution in volunteer clouds, as ants' decisions do not take into account the QoS requirements of the tasks.

Load balancing ant colony optimization (LBACO) [31] is an extension of the basic ACO algorithm of Di Caro and Dorigo [9]. LBACO tries to find the optimal resource allocation for each task, and to minimize the *makespan* of a given task set, adapting to the dynamic cloud computing system and balancing the entire system load. The makespan is defined as the time difference among the task that completes first and the one that completes last. The basic ACO algorithm is extended by carrying out scheduling decisions that take into account the results of previous ones, and also considering the load of each VM. The algorithm takes into account VM features such as the number of available processors, its million instructions per second (MIPS) capability, and communication bandwidth. LBACO is evaluated through simulation and compared to basic first-in, first-out (FIFO) and ACO algorithms, in terms of average makespan and degree of imbalance (a measure of imbalance among VMs). The ACO algorithm considered in the present work instead has a different purpose, as it considers only individual tasks, which have an associated deadline parameter, and tries to maximize the number of completed tasks while respecting their QoS requirements. LBACO cannot be directly applied to collaborative task execution in volunteer clouds, as it assumes that each node knows all resources available in the neighbor nodes, which is unrealistic in those scenarios.

The idea of colored ants was previously presented in a completely different way by Ali and Belal [2]. They considered a multiple colony approach, where each node sends a colored colony throughout the network. Colored ant colonies help in preventing ants of the same nest from following the same route, hence enforcing them to be distributed all over the nodes in the network. One main difference with respect to our work is that Ali and Belal's ants tend to maximize the coverage of the network (exploration), whereas our scout ants can be configured with a certain exploration-exploitation trade-off, according to the softmax method (see Equation (1)).

A different approach was adopted by Di Nitto et al. [37], who used bioinspired algorithms to balance the workload (i.e., to ensure that all nodes have almost the same amount of tasks in their queue). To deal with node heterogeneity, the algorithm proceeds in two steps: first, the network is rewritten to cluster the nodes of the same type; second, messages are spread among nodes of the same type, to redistribute the load. The goal of such an algorithm is different from ours. Moreover, we consider nodes with heterogeneous resource characteristics, not belonging to only a few classes.

5.2 Cloud Partitioning

Geographical cloud partitioning has already been adopted by many public cloud providers. Multiple geographical locations can provide redundancy and ensure reliability in the event of site failures.

A game-theoretical model for load balancing in partitioned clouds was proposed by Xu et al. [64]. There, the cloud is partitioned according to the data center's geographical location. A centralized

node acts as load balancer receiving and dispatching the jobs. Each machine periodically provides information about its characteristics and load to the central node. If an execution request cannot be satisfied, the load balancer forwards the request to another site according to a round-robin approach after having ordered the queue with updated information on the load of nodes.

The authors of Khare and Chauhan [27] reviewed several load balancing implementation works where clouds are partitioned. Disregarding the implementation details in a real public cloud, which are outside the scope of the presented work, all reviewed architectures work with geographical partitions with the rationale of minimizing the network overhead due to the transmission of load information.

A mathematical formulation of system performance prediction in the cloud, based on queueing theory, was studied in Pacheco-Sanchez et al. [39]. There, a Markovian arrival process was used as a tool for characterizing workload fluctuations under time-varying traffic intensities. Such a prediction model can be implemented in the supernodes while tuning the size of cloud sites.

Load balancing can be exploited even from an energy efficiency point of view. Adnan et al. [1] proposed a geographical load balancing where the central dispatcher node considers workload latency requirements and electricity price variations to choose how much workload should be executed in each data center to obtain cost savings.

5.3 Cloud Simulators

In its early days, cloud systems were mainly simulated by means of the already available grid computing simulators. The huge popularity gained by cloud computing in the subsequent years encouraged the development of new and more specific simulators, such as CloudSim [7], GreenCloud [29], iCanCloud [38], and GloudSim [13]. For a detailed discussion of the relation with our simulator, we refer to Sebastio et al. [45]. We just mention here that a direct point-to-point comparison among the mentioned simulators and AVoCloudy cannot be done, as their purposes are very different: either to accurately simulate few interconnected data centers focusing on network aspects or to study the cost-performance trade-off or the job's life cycle.

6 CONCLUSION AND FUTURE WORK

Volunteer cloud computing can help overcome the scalability limitations of cloud computing by harnessing the users' willingness to share the spare resources of their machines. The complexity, dynamism, and heterogeneity of volunteer cloud computing systems pose several challenges to their engineering and prediction.

We have presented a novel holistic approach in which intelligent agents in the overlay and in the application execution layers of all cloud participants cooperate to provide a collective task management service. We have shown an instantiation of our approach through a novel combination of two promising techniques: ACO of task scheduling [46] and overlay network partitioning [49]. The first one is a self-adaptive, highly parametric algorithm suited for collaborative task execution problems, relying on the computational field framework inspired by ACO [16] and spatial computing [65] approaches. The latter one is a dynamic workload-based partitioning of the overlay network, with the aim of supporting the task scheduling algorithm running on top of it.

The performance of the proposed approach has been evaluated by means of simulation-based statistical analysis using a compute-intensive workload from Google [22, 32] and comparing it to other known algorithms. The results show that the proposed holistic approach outperforms all other techniques in terms of perceived QoS because of its collective, decentralized, collaborative, and self-adaptive nature.

In the proposed approach, adaptiveness resides both at the overlay layer (i.e., when scout ants maintain the network) and at the application execution layer (i.e., when the partitioning is adapted

to the load of the nodes and the task characteristics). In future work, we plan to incorporate further adaptivity capabilities, such as tuning the parameters that control the ACO-based approach. At the application execution layer, we plan to evaluate further features of our algorithm, with particular attention to the self-adaptive antistagnation mechanisms proposed in Sebastio et al. [46] (i.e., angry ants and memory aging). Moreover, we plan to investigate novel mechanisms based on heterogeneous ants (i.e., ants that have different behaviors), as well as to study variants of the basic spatial computing based on the information diffusion rules that we have used in the experiments. Furthermore, lightweight performance monitoring and workload classification mechanisms could be implemented. Workload classification could be used by agents to dynamically adjust the number and size of each cloud site. Moreover, additional experiments could evaluate the system performance when agents adopt a multicriteria optimization (i.e., assign priority to more than one metric) while tuning the partitions (e.g., the queue waiting time for small tasks can be more important than the number of successfully completed large tasks). Since overlay partitioning is suitable to be used with any workload, we will also plan to study the performance of our algorithms under additional workload and traffic models (e.g., Kim et al. [28]; Lawrence Berkeley National Laboratory [30]; Hebrew University, Experimental Systems Lab [56]; TU Delft, Delft University of Technology [58]), in which most likely a different number of classes of tasks is present.

ACKNOWLEDGMENTS

Antonio Scala thanks CNR-PNR National Project Crisis-Lab for support. M. Amoretti was supported by the University of Parma Research Fund–FIL 2016–Project "NEXTALGO: Efficient Algorithms for Next-Generation Distributed Systems." The contents of the article do not necessarily reflect the position or the policy of funding parties.

REFERENCES

[1] Muhammad Abdullah Adnan, Ryo Sugihara, and Rajesh K. Gupta. 2012. Energy efficient geographical load balancing via dynamic deferral of workload. In *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD'12)*. IEEE, Los Alamitos, CA, 188–195. DOI: http://dx.doi.org/10.1109/CLOUD.2012.45

[2] Al-Dahoud Ali and Mohamed A. Belal. 2007. Multiple ant colonies optimization for load balancing in distributed systems. In *Proceedings of the 2007 1st International Conference on Information and Communication Technology and Accessibility (ICTA'07)*.

[3] Michele Amoretti, Alberto Lluch Lafuente, and Stefano Sebastio. 2013. 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'13)*. 274–281. DOI: http://dx.doi.org/10.1109/PDP.2013.47

[4] Michele Amoretti, Marco Picone, Francesco Zanichelli, and Gianluigi Ferrari. 2013. Simulating mobile and distributed systems with DEUS and ns-3. In *Proceedings of the 2013 International Conference on High Performance Computing Simulation (HPCS'13)*. 107–114. DOI: http://dx.doi.org/10.1109/HPCSim.2013.6641400

[5] Ozalp Babaoglu and Moreno Marzolla. 2014. The people's cloud. *IEEE Spectrum* 51, 10, 50–55.

[6] Ozalp Babaoglu, Moreno Marzolla, and Michele Tamburini. 2012. Design and implementation of a P2P cloud system. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC'12)*. ACM, New York, NY, 412–417. DOI: http://dx.doi.org/10.1145/2245276.2245357

[7] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience* 41, 1, 23–50. DOI: http://dx.doi.org/10.1002/spe.995

[8] Justin Cappos. 2016. Seattle: Open Peer-to-Peer Computing Home Page. Retrieved February 5, 2018, from https://seattle.poly.edu/html/.

[9] Gianni Di Caro and Marco Dorigo. 1997. *AntNet: A Mobile Agents Approach to Adaptive Routing*. Technical Report. IRIDIA.

[10] Simon Caton and Omer Rana. 2012. Towards autonomic management for cloud services based upon volunteered resources. *Concurrency and Computation: Practice and Experience* 24, 9, 992–1014. DOI: http://dx.doi.org/10.1002/cpe.1715.

[11] Alessandro Celestini, Alberto Lluch Lafuente, Philip Mayer, Stefano Sebastio, and Francesco Tiezzi. 2014. Reputation-based cooperation in the clouds. In *Trust Management VIII*. IFIP Advances in Information and Communication Technology, Vol. 430. Springer, 213–220. DOI: http://dx.doi.org/10.1007/978-3-662-43813-8_15

[12] Fernando Costa, Luis Silva, and Michael Dahlin. 2011. 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'11)*. 1855–1862. DOI: http://dx.doi.org/10.1109/IPDPS.2011.345

[13] Sheng Di and Franck Cappello. 2015. GloudSim: Google trace based cloud simulator with virtual machines. *Software: Practice and Experience* 45, 11, 1571–1590. DOI: http://dx.doi.org/10.1002/spe.2303

[14] University of Parma Distributed Systems Group. 2009. DEUS. Retrieved February 5, 2018, from https://github.com/dsg-unipr/deus/.

[15] Wenrui Dong, Guangming Liu, Jie Yu, and You Zuo. 2015. Characterizing I/O workloads of HPC applications through online analysis. In *Proceedings of the 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC'15)*. 1–2. DOI: http://dx.doi.org/10.1109/PCCC.2015.7410353

[16] Marco Dorigo and Luca M. Gambardella. 1997. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1, 1, 53–66. DOI: http://dx.doi.org/10.1109/4235.585892

[17] EDGI Consortium. 2012. European Project RI-261556 EDGI (European Desktop Grid Initiative). Retrieved February 5, 2018, from http://edgi-project.eu/.

[18] Adriano Fiorese, Paulo Simões, and Fernando Boavida. 2012. Peer selection in P2P service overlays using geographical location criteria. In *Computational Science and Its Applications—ICCSA 2012*. Lecture Notes in Computer Science, Vol. 7334. Springer, 234–248. DOI: http://dx.doi.org/10.1007/978-3-642-31075-1_18

[19] Toktam Ghafarian, Hossein Deldari, Bahman Javadi, and Rajkumar Buyya. 2013. A proximity-aware load balancing in peer-to-peer-based volunteer computing systems. *Journal of Supercomputing* 65, 797–822. DOI: http://dx.doi.org/10.1007/s11227-012-0866-7

[20] Toktam Ghafarian, Hossein Deldari, Bahman Javadi, Mohammad H. Yaghmaee, and Rajkumar Buyya. 2013. CycloidGrid: A proximity-aware P2P-based resource discovery architecture in volunteer computing systems. *Future Generation Computer Systems* 29, 6, 1583–1595. DOI: http://dx.doi.org/10.1016/j.future.2012.08.010

[21] Li Gong. 2001. JXTA: A network programming environment. *IEEE Internet Computing* 5, 3, 88–95. DOI: http://dx.doi.org/10.1109/4236.935182

[22] Joseph L. Hellerstein. 2010. Google cluster data. *Google Research Blog*. Retrieved February 5, 2018, from https://research.googleblog.com/2010/01/google-cluster-data.html.

[23] Youki Kadobayashi. 2004. Achieving heterogeneity and fairness in Kademlia. In *Proceedings of the 2004 International Symposium on Applications and the Internet Workshops (SAINT Workshops'04)*. 546–551. DOI: http://dx.doi.org/10.1109/SAINTW.2004.1268686

[24] Hanna Kavalionak, Emanuele Carlini, Laura Ricci, Alberto Montresor, and Massimo Coppola. 2015. Integrating peer-to-peer and cloud computing for massively multiuser online games. *Peer-to-Peer Networking and Applications* 2, 1–19. DOI: http://dx.doi.org/10.1007/s12083-013-0232-4

[25] Hanna Kavalionak and Alberto Montresor. 2012. P2P and cloud: A marriage of convenience for replica management. In *Self-Organizing Systems*. Lecture Notes in Computer Science, Vol. 7166. Springer, 60–71. DOI: http://dx.doi.org/10.1007/978-3-642-28583-7_6

[26] Jeffrey O. Kephart and David M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1, 41–50. DOI: http://dx.doi.org/10.1109/MC.2003.1160055

[27] Suchita Khare and Abhishek Chauhan. 2014. A review on load balancing model based on cloud partitioning for the public cloud. *International Journal of Emerging Technology and Advanced Engineering* 4, 7, 736–744.

[28] Wonho Kim, Ajay Roopakalu, Katherine Y. Li, and Vivek S. Pai. 2011. Understanding and characterizing PlanetLab resource usage for federated network testbeds. In *Proceedings of the 2011 ACM SIGCOMM Internet Measurement Conference (IMC'11)*. ACM, New York, NY, 515–532. DOI: http://dx.doi.org/10.1145/2068816.2068864

[29] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. 2012. GreenCloud: A packet-level simulator of energy-aware cloud computing data centers.*Journal of Supercomputing* 62, 3, 1263–1283. DOI: http://dx.doi.org/10.1007/s11227-010-0504-1

[30] Lawrence Berkeley National Laboratory. 2008. The Internet Traffic Archive. Retrieved February 5, 2018, from http://ita.ee.lbl.gov.

[31] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, and Dan Wang. 2011. Cloud task scheduling based on load balancing ant colony optimization. In *Proceedings of the 2011 6th Annual Chinagrid Conference*. 3–9. DOI: http://dx.doi.org/10.1109/ChinaGrid.2011.17

[32] Asit K. Mishra, Joseph L. Hellerstein, Walfredo Cirne, and Chita R. Das. 2010. Towards characterizing cloud backend workloads: Insights from Google compute clusters. *ACM SIGMETRICS Performance Evaluation Review* 37, 4, 34–41. DOI: http://dx.doi.org/10.1145/1773394.1773400

[33] Ratan Mishra and Anant Jaiswal. 2012. Ant colony optimization: A solution of load balancing in cloud. *International Journal of Web and Semantic Technology* 3, 2, 33–50. DOI: http://dx.doi.org/10.5121/ijwest.201203203

[34] Alberto Montresor and Luca Abeni. 2011. Cloudy weather for P2P, with a chance of gossip. In *Proceedings of the 2011 IEEE International Conference on Peer-to-Peer Computing (P2P'11)*. 250–259. DOI: http://dx.doi.org/10.1109/P2P.2011.6038743

[35] Alessandro Moro, Enzo Mumolo, and Massimiliano Nolich. 2009. Ergodic continuous hidden Markov models for workload characterization. In *Proceedings of 6th International Symposium on Image and Signal Processing and Analysis*. 99–104. DOI: http://dx.doi.org/10.1109/ISPA.2009.5297771

[36] NaDa. 2008. European Integrated Project 223850 NaDa (Nano Data Centers). Retrieved February 9, 2015, from https://cordis.europa.eu/project/rcn/86610_en.html.

[37] Elisabetta Di Nitto, Daniel J. Dubois, and Raffaella Mirandola. 2009. 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*. 68–75. DOI: http://dx.doi.org/10.1109/SEAMS.2009.5069075

[38] Alberto Núñez, Jose L. Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero, and Ignacio M. Llorente. 2012. iCanCloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing* 10, 1, 185–209. DOI: http://dx.doi.org/10.1007/s10723-012-9208-5

[39] S. Pacheco-Sanchez, G. Casale, B. Scotney, S. McClean, G. Parr, and S. Dawson. 2011. Markovian workload characterization for QoS prediction in the cloud. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD'11)*. 147–154. DOI: http://dx.doi.org/10.1109/CLOUD.2011.100

[40] Danilo Pianini, Stefano Sebastio, and Andrea Vandin. 2014. Distributed statistical analysis of complex systems modeled through a chemical metaphor. In *Proceedings of the 2014 International Conference on High Performance Computing Simulation (HPCS'14)*. 416–423. DOI: http://dx.doi.org/10.1109/HPCSim.2014.6903715

[41] Stuart J. Russell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach* (2nd ed.). Pearson Education.

[42] Lorenzo Saino, Cosmin Cocora, and George Pavlou. 2013. A toolchain for simplifying network simulation setup. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques (SimuTools'13)*. 82–91.

[43] Stefano Sebastio. 2014. AVoCloudy. Retrieved February 5, 2018, from https://github.com/distributedResearch/avocloudy.

[44] Stefano Sebastio. 2014. *Enriching Volunteer Couds With Self-* Capabilities*. Ph.D. Dissertation. IMT School for Advanced Studies Lucca. http://e-theses.imtlucca.it/146/.

[45] Stefano Sebastio, Michele Amoretti, and Alberto Lluch Lafuente. 2016. AVOCLOUDY: A simulator of volunteer clouds. *Software: Practice and Experience* 46, 1, 3–30. DOI: http://dx.doi.org/10.1002/spe.2345

[46] Stefano Sebastio, Michele Amoretti, and Alberto Lluch Lafuente. 2014. 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'14)*. ACM, New York, NY, 105–114. DOI: http://dx.doi.org/10.1145/2593929.2593943

[47] Stefano Sebastio and Giorgio Gnecco. 2017. A green policy to schedule tasks in a distributed cloud. *Optimization Letters*. Available at https://link.springer.com/journal/11590 DOI: http://dx.doi.org/10.1007/s11590-017-1208-8

[48] Stefano Sebastio, Giorgio Gnecco, and Alberto Bemporad. 2017. Optimal distributed task scheduling in volunteer clouds. *Computers and Operations Research* 81, 231–246. DOI: http://dx.doi.org/10.1016/j.cor.2016.11.004

[49] Stefano Sebastio and Antonio Scala. 2015. A workload-based approach to partition the volunteer cloud. In *Proceedings of the 2015 IEEE Conference on Collaboration and Internet Computing (CIC'15)*. 210–218. DOI: http://dx.doi.org/10.1109/CIC.2015.27

[50] Stefano Sebastio and Andrea Vandin. 2013. MultiVeStA: Statistical model checking for discrete event simulators. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools (ValueTools'13)*. DOI: http://dx.doi.org/10.4108/icst.valuetools.2013.254377

[51] Ben Segal, Predrag Buncic, David Garcia Quintas, Daniel Lombrana Gonzales, Artem Harutyunyan, Jarno Rantala, and David Weir. 2009. *Building a Volunteer Cloud*. Technical Report. CERN. http://ben.web.cern.ch/ben/Ven_abs.pdf.

[52] Kwang Mong Sim and Weng Hong Sun. 2003. Ant colony optimization for routing and load-balancing: Survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans* 33, 5, 560–572. DOI: http://dx.doi.org/10.1109/TSMCA.2003.817391

[53] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for Internet applications. *ACM SIGCOMM Computer Communication Review* 31, 4, 149–160. DOI: http://dx.doi.org/10.1145/964723.383071

[54] Daniel Stutzbach and Reza Rejaie. 2006. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference (IMC'06)*. ACM, New York, NY, 189–202. DOI: http://dx.doi.org/10.1145/1177080.1177105

[55] Domenico Talia. 2011. Cloud computing and software agents: Towards cloud intelligent services. In *Proceedings of the 12th Workshop on Objects and Agents (WOA'11)*. 2–6.

[56] Hebrew University, Experimental Systems Lab. 2015. Logs of Real Parallel Workloads From Production Systems. Retrieved February 5, 2018, from http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.
[57] Trustees of Princeton University. 2015. PlanetLab Home Page. Retrieved February 5, 2018, from https://www.planet-lab.org/.
[58] TU Delft, Delft University of Technology. 2010. The Grid Workloads Archive. Retrieved February 5, 2018, from http://gwa.ewi.tudelft.nl/.
[59] Jeffrey D. Ullman. 1975. NP-complete scheduling problems. *Journal of Computer and System Sciences* 10, 3, 384–393. DOI: http://dx.doi.org/10.1016/S0022-0000(75)80008-0
[60] University of California. 2017. BOINC Home Page. Retrieved February 5, 2018, from http://boinc.berkeley.edu/.
[61] University of California, Berkeley. 2016. SETI@home Home Page. Retrieved February 5, 2018, from http://setiathome.ssl.berkeley.edu/.
[62] University of Wisconsin-Madison. 2017. HTCondor Home Page. Retrieved February 5, 2018, from http://research.cs.wise.edu/htcondor/.
[63] Andrea Vandin and Stefano Sebastio. 2013. MultiVeStA. Retrieved February 5, 2018, from http://sysma.imtlucca.it/tools/multivesta.
[64] Gaochao Xu, Junjie Pang, and Xiaodong Fu. 2013. A load balancing model based on cloud partitioning for the public cloud. *Tsinghua Science and Technology* 18, 1, 34–39. DOI: http://dx.doi.org/10.1109/TST.2013.6449405
[65] Franco Zambonelli and Marco Mamei. 2005. *Spatial Computing: An Emerging Paradigm for Autonomic Computing and Communication*. Springer, Berlin, Germany, 44–57. DOI: http://dx.doi.org/10.1007/11520184_4

Received January 2017; revised July 2017; accepted October 2017