A Computational Field Framework for Collaborative Task Execution in Volunteer Clouds

Stefano Sebastio
IMT Institute for Advanced Studies
Lucca, Italy
stefano.sebastio@imtlucca.it

Michele Amoretti
Centro Interdipartimentale SITEIA.PARMA,
Università degli Studi di Parma, Italy
michele.amoretti@unipr.it

Alberto Lluch Lafuente
IMT Institute for Advanced Studies
Lucca, Italy
alberto.lluch@imtlucca.it

Abstract—**The increasing diffusion of the cloud computing paradigm is opening new opportunities for distributed and collaborative computing. Volunteer clouds are a prominent example, where participants join and leave the platform and collaborate by sharing their computational resources. The high dynamism and unpredictability of such scenarios call for decentralized self-* approaches to guarantee QoS. We present a simulation framework for collaborative task execution in volunteer clouds and propose one instance of the framework based on Ant Colony Optimization, that we validate through simulation-based statistical analysis over Google workload data.**

Keywords*—*cloud computing; volunteer computing; distributed tasks execution; ant colony optimization; bio-inspired algorithms; spatial computing; peer-to-peer.

I. Introduction

The wide adoption of the cloud computing paradigm is increasing the efforts of the research community on approaches and techniques to optimize resource allocation. Usually, cloud service providers arrange their resources in sites that cooperate within the domain of the same company. However, new peer-to-peer (P2P), decentralized, open-world paradigms such as *Volunteer Computing* [1] are gaining popularity. Such paradigms envision platforms where, in addition to data centers, less powerful computational devices participate to share and use each others' resources, and are characterized by a high, unpredictable dynamism (participants may leave and join at any time) and heterogeneity (participants may share and need different computational resources). Since global coordination and optimization techniques can be hardly applied, the attention has been shifted to the application of agent-based techniques to cloud computing (e.g. as advocated in [2]), like Ant Colony Optimization (ACO) [3] and Spatial Computing [4]. Such approaches provide flexible and scalable solutions to distributed computing problems, such as collaborative task execution.

We present here a framework designing and evaluating collaborative task execution algorithms for volunteer cloud computing platforms. The framework can be seen as an evolution of our preliminary work [5], that we extend here in several directions. First, we introduce in §II a distributed data structure — called *Colored Computational Field* — inspired by spatial fields, routing tables and ACO's pheromone-based stigmergy, which provides a suitable basis for many agent-based collaborative task execution algorithms. Second, by means of the aforementioned framework, we define in §III a highly parametric ACO-based algorithm, offering a decentralized solution characterized by lightweight ant agents (in terms of behavior and carried knowledge), which maintain and exploit the colored computational field and do not require any additional data structures. In §IV we report an excerpt of the experimental evaluation of the ACO-based algorithm, where we assess the performance of various alternatives and parameters, using the workload described by the Google Cluster dataset [6]. In §V we discuss the main sources of inspiration and further related work. Finally, in §VI we provide some concluding remarks and outline our current and future research efforts.

All in all, our work provides (i) a flexible framework where existing or new agent-based algorithms for collaborative cloud computing problems can be designed and evaluated; (ii) a novel, highly parametric ACO-based algorithm, which we advocate as a strong candidate for collaborative task execution problems.

II. Colored Computational Field

We consider a *Volunteer Cloud* as a network of participants (also called nodes) equipped with a set $\mathcal{R}$ of computational resources, which can enter and leave the system any time, and submit and satisfy task execution requests, subject to QoS requirements. When a node is not able to execute its own tasks, it needs to find another node able to do it. Such a search must take into account that, as in social networks, the node's visibility is restricted to its contacts up to a certain degree, and, at the same time, the amount of spent time and messages spread in the network should be minimized.

The main supporting structure of our framework is a *Colored Computational Field*, used to facilitate the discovery of nodes which can satisfy task execution requests.

Definition 2.1 (Colored Computational Field): Let K be a set of $\mathbb{R}^+$-valued computational pheromones. A *K-colored computational field* is a tuple $\langle N, E, \rho, \Phi\rangle$ such that N is a set of nodes (representing cloud participants), $E \subseteq N \times N$ is a set of edges (representing contact relations among participants), $\rho : N \to (\mathbb{R}^+)^{|\mathcal{R}|}$ is a *resource map* (i.e., a mapping of nodes to their computational resources), and $\Phi : E \to [0, 1]^{|K|}$ is the *pheromone table* of each edge.

Usually, $\mathcal{R} \subseteq K$, i.e., each element of K corresponds to a computational resource (e.g., memory amount, number of cores, core frequency), but it may also contain other values. We shall consider, for instance, the *predicted idle time* and a *feedback pheromone* as elements of K in our examples. For the sake of simplicity we assume that all nodes feature the same set of resources and that those are measurable in

$\mathbb{R}^+$ (i.e., as non-negative reals). We denote the subset of QoS requirements of a task t which regard the computational resources $\mathcal{R}$ by t_Q, modeled as a vector in $(\mathbb{R}^+)^{|\mathcal{R}|}$, to be interpreted as lower bounds on each resource. Additional requirements such as deadlines are treated separately. The resource map ρ is used to represent each node's computational resources. The pheromone table Φ is a mapping of edges into a vector of $[0,1]$-normalized pheromone values. Each value in the vector is a "pheromone level" value associated to one of the K computational pheromones and indicates a sort of level of "goodness" of a connection with respect to the resource. Obviously, Φ is intended to be implemented as a distributed table where each entry $\Phi(i,j)$ is maintained at node i. We assume that both $\mathcal{R}$ and K are ordered sets so that we can refer to the individual components in the vectors returned by ρ and Φ by their position. Indeed, we often refer to the pheromone k in edge e_j with $\Phi_k(e_j)$. More in general we denote with $\vec{x}_i$ the i-th element of a vector $\vec{x}$. We sometimes refer to the set of edges (i,j) outgoing from a node i with E_i. The pheromone table can be seen as a sort of routing table or *gradient map* [4], used to ease resource discovery while minimizing communication.

III. ACO-BASED ALGORITHM

Several algorithms can be defined on top of a Colored Computational Field, including those based on local diffusion rules typical of spatial computing approaches [4]. This section presents a paradigmatic and, at the same time, novel example of a highly parametric ACO-based algorithm. Contrary to typical spatial computing approaches the algorithm uses ant-like mobile agents that are in charge of maintaing and exploiting the Colored Computational Field. This algorithm relies on two different types of ants: *colored scout ants* and *hunter ants*. Colored scout ants periodically explore the neighborhood of a node to discover computational resources, to update the field accordingly. Such ants are specialized by computational pheromones: each color k corresponds to one of the computational pheromones in K. Hunter ants are spawned when a task execution request is issued. They exploit the field to find a volunteer node, and update the field according to the received feedback.

Both types of ants are described in detail in §III-A and §III-B, respectively. It is worth to remark a key feature they have in common: both exploit the field to take their exploration decisions, namely when they are in a node they choose their next hop with a probabilistic selection weighted according to the corresponding level of pheromone. Such an operation, called *stigmergy*, may eventually lead to an optimal situation in a static network, but may also suffer (as all ACO-based approaches) from *stagnation*, specially in dynamic networks. Stagnation occurs when the ants converge to an apparently optimal decision, which may prevent the system to adapt to the emergence of new, 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 invers proportion to the distance to resources) and *angry ants* (a third kind of agents that remove pheromones along outdated links).

A. Colored Scout Ants

Colored scout ants are periodically spawned in a process that is independent from the request and execution of tasks. Their goal is to explore the network and update the pheromone field. Each ant releases and follows its own pheromone color ($k \in K$). There may be computational colors with no associated colored scout ants. In our case, for instance, no scout is in charge of the feedback pheromone since this is to be maintained by the hunter ants. Listing 1 describes the behavior of scout ants through a pseudo-code. To summarize, each scout ant explores the network (line 7), probing the neighborhood goodness while going away from its home node (the one that has spawned it). Each ant has an associated time-to-live (TTL), which establishes the number of hops an ant must try to do during its exploration, before returning home — that prevents endless and unnecessary exploration efforts. When its TTL is exhausted, the scout ant comes back (line 17) to its source node releasing the pheromone according to the memory aging approach (line 31). We provide a detailed explanation of the main features of the algorithm.

Temperature-dependent Exploration & Exploitation. The behavior of ants is based on online Reinforcement Learning (RL) [7], where at each step the decision involves a choice among: *exploration* (try to gather new information) and *exploitation* (focus on the best decision according the current information). Exploration can 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 [7]. Each ant moves according to the past path desirability (exploitation) and to the exploration compliance, according to the following equation:

$$p_k(e_j) = \frac{e^{\frac{\Phi_k(e_j)}{T_i}}}{\sum_{\forall e_q \epsilon E_i} e^{\frac{\Phi_k(e_q)}{T_i}}} \quad (1)$$

that defined the probability $p_k(e_j)$ that the k-colored scout ant at node i chooses e_j as the next hop. According to the Softmax action selection method, we have chosen the Boltzmann/Gibbs distribution, with a tunable temperature function T_i, to choose the next hop from node i probabilistically, but 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 tradeoff, i.e., if $T_i \to \infty$ the ant at node i tends to follow a more random approach (all the paths have the same preference), otherwise if $T_i \to 0$ the ant follows a greedy approach, which reduces the exploration component.

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, each time a new neighbor connects to a node i, the corresponding function T_i is re-initialized to encourage the exploration of new resources.

Memory Aging. Scout ants explore the network and record the *nodes' 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 ruled by the ant memory aging factor (to prevent stagnation) and the node goodness in that part of the network (Listing 1, lines

Listing 1. Colored scout ant algorithm

```
1  coloredAntStep(ScoutAnt ant_k){
2    ant_k.pathAdd(this);
3    ant_k.pathNest(this.getNestGoodness(k));
4    ant_k.updateTtl();
5
6    if (ant_k.getTtlValue()>0){
7      w := antChooseContact(this.neighbors - ant_k.getPath());
8      if (w!=0){
9        w.coloredAntStep(ant_k);
10       return;
11     }
12   }
13   l := ant_k.getStepPrevious(this);
14   l.coloredAntStepBack(ant_k, this);
15 }
16
17 coloredAntStepBack(ScoutAnt ant_k, Node from){
18   this.depositColoredPheromone(ant_k, from);
19   l := ant_k.getStepPrevious(this);
20   l.coloredAntStepBack(ant_k, this);
21 }
22
23 depositColoredPheromone(ScoutAnt ant_k, Node from){
24   p := ant_k.getMemoryAgingPheromone(this);
25   if ( (p > this.getPheromoneEdge(from)) ||
26     (k != FINISHING_TIME )){
27      this.setPhermoneEdge(p, from);
28     }
29 }
30
31 getMemoryAgingPheromone(Node n){
32   antMemory_trace = pathNest.subList(n.index+1, end);
33   p_best = max(antMemory_trace);
34   mem_aging =|antMemory_trace.getIndex(p_best)-(n.index+1)|;
35   return p_memoryAging(p_best, mem_aging);
36 }
```

31-36, getMemoryAgingPheromone(·)). We do not use the traditional concept of aging, where the ants deposit lesser and lesser pheromone as they moves from node to node, because the information that the pheromone provides, in our setting, is useful not only for the node where the scout has been spawned. However, we still want to take into account the distance between a potential task execution requester and the node holding the necessary resources. For this purpose, our memory aging mechanism releases an amount of pheromone that is inversely proportional to the distance to the best resource found so far, and not to the distance between the node from which the ant has been spawned (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 p_memoryAging(p_best, mem_aging), illustrated in Listing 1 at line 35 as $\texttt{p\_best} - \texttt{mem\_aging} \cdot \texttt{AgingFactor}$, where p_best is the best value found so far, mem_aging is the distance to it and AgingFactor is a discounting factor.

Fig. 1 clarifies the memory aging approach through an example. The scout ant is spawned at node 0 and follows the path $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ (Fig. 1, top). When the TTL expires after 5 hops (at node 5) the scout ant returns home (node 0). In the first step back the actual value of the resource at node 5 (i.e., 3) is taken into account (see the label on the link from node 4 to 3). Note, however, that in the second step back the edge is labelled with 2.5 and not 3, as an effect of the aging function. At each step back, the pheromone on the next link is updated only if its value is lower than the one the ant would like to assign. Otherwise the current value is kept.

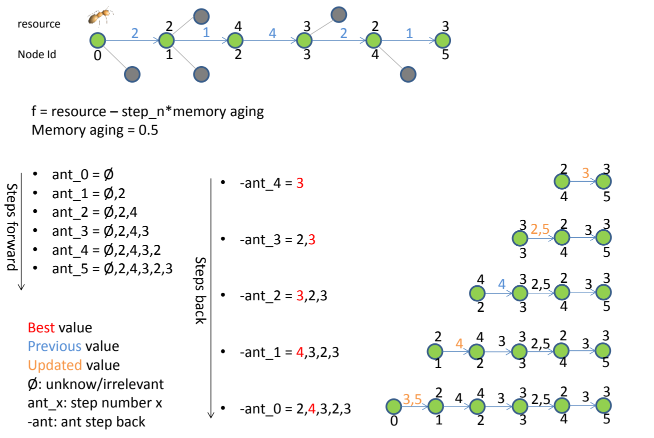

Fig. 1. Example of memory aging approach for scout ants.

This is the case of the third step back (from node 3 to node 2), where the ant has found a resource with value 3 but the previous pheromone value is 4 (that may be the result from a previous exploration of some of the subsequent gray nodes).

Evaporation. In addition to dynamic temperature and memory aging mechanisms, we also use the evaporation technique to deal with stagnation in 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. A new pheromone is released only if the new value is higher than the one previously released. If instead we would update the pheromone values regardless of the best previously found values, they would be highly variable, providing unstable information. We consider resources such as the amount of RAM and CPU characteristics to be non-volatile, since those cannot be allocated forever but only on short-basis (i.e., to execute tasks). Thus, until the node participates to 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, the former uses angry ants (described below) to update the field.

Angry Ants. Despite the non-volatile nature of resources, the instable nature of the network of participants [8] 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 of 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.

B. Hunter Ants

When a node has a task for which it cannot respect the deadline, it starts spawning multiple *hunter ants*. Every hunter ant tries to find a node ready to satisfy the task execution request.

Listing 2. Hunter ant algorithm

```
1  antStep(Ant ant){
2    ant.pathAdd(this);
3    ant.updateTtl();
4    if (this.askExecutionToNode(ant.getTask())){
5      l := ant.getStepPrevious(this);
6      l.antStepBack(ant, this);
7    } else if (ant.getTtlValue()>0) {
8      w := antChooseContact(this.neighbors - ant.getPath());
9      if (w!=0){
10       w.antStep(ant);
11       return;
12     }
13   }
14   this.antStepBackHome(ant);
15 }
16
17 antStepBack(Ant ant, Node from){
18   this.depositPheromone(ant, from);
19   l := ant.getStepPrevious(this);
20   l.antStepBack(ant, this);
21 }
22
23 depositPheromone(Ant ant, Node from){
24   p := ant.getAgingPheromone(this);
25   this.setPhermoneEdge(p, from);
26 }
```

For this purpose the hunter ant starts exploring the network, exploiting the field and the task characteristics. Task execution requests are sent to nodes 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 not functional requirements) and not the task itself, to minimize transmission overheads.

The behavior of hunter ants is sketched in Listing 2, where the ant contains a task description used to find the best match (as we shall explain). Each hunter ant tries to find a node willing to execute the task (line 4), following the Colored Computational Field (line 8) built according to the overall pheromone — see below Eq. 3. If the hunter ant does not find any node willing to collaborate after its TTL, it returns to the home node. In the following we provide a detailed explanation of the main features of the algorithm.

Resource Allocation Heuristic. The global goal of the system would be to maximize the number of tasks that meet their deadline. However, the problem is clearly untractable in a global manner (for instance, even the problem of finding the best task-node match is well known to be *NP-complete*) and would require perfect predictions of future task arrival times and characteristics, which is totally unrealistic in open environment such as volunteer clouds, 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 used completely) will increase the probability to accommodate more requests in the future. These heuristics rely on a *single resource waste ratio* function $srwr$ and a *combined resource waste ratio* function $crwr$, defined in Eq. 2. Note that the latter uses a vector η of size $|\vec{x}|$, which allows one to express preferences among resources.

$$srwr(x,y) = \frac{min(x,y)}{max(x,y)} \qquad crwr(\vec{x},\vec{y}) = \sum_{\forall k \in 1..|\vec{x}|} \frac{\eta_k \cdot srwr(\vec{x}_k, \vec{y}_k)}{\sum_{\forall \sigma \in 1..|\vec{x}|} \eta_\sigma} \tag{2}$$

TABLE I. Example of Resources under/over utilization

$\vec{x}$	$\vec{y}$	$srwr(\vec{x}_1,\vec{y}_1)$	$srwr(\vec{x}_2,\vec{y}_2)$	$crwr(\vec{x},\vec{y})$
$\langle M,N\rangle$	$\langle M,N\rangle$	1	1	1
$\langle M/2,N\rangle$	$\langle M,N\rangle$	0.5	1	0.75
$\langle M/2,N/2\rangle$	$\langle M,N\rangle$	0.5	0.5	0.5
$\langle 0,N\rangle$	$\langle M,N\rangle$	0	1	0.5

These functions are exemplified in Table I, where two types of resources are taken into account, both with the same weight ($\eta_1 = \eta_2 = 1$). The first example is the best match, where the required resources $\vec{x}$ perfectly match the provided resources $\vec{y}$. The rest of the cases exemplify mismatches due to under/over resource utilization. Smaller values of $crwr$ suggest higher degree of mismatch between requested and provided resources.

Weighting links. Such heuristic functions are used to associate goodness values to links. For this purpose we also use a function $\Phi(t_{\mathcal{Q}})$ which provides the pheromone vector for the resources required by a task t obtained by applying the same functions used by scout ants. Then the goodness of a link e will be based on the value of $crwr(\Phi_{\mathcal{R}}(e), \Phi(t_{\mathcal{Q}}))$, where $\Phi_{\mathcal{R}}(e)$ is the pheromone vector associated to all computational resources in $\mathcal{R}$ (which coincide with those expressed in task's QoS). 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, while 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 ${}^{\Phi_k(t_{\mathcal{Q}})}/{}_{\Phi_k(e)}$ when the resource is under-used, or ${}^{\Phi_k(e)}/{}_{\Phi_k(t_{\mathcal{Q}})}$ when the resource is over-used.

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 network stability and also its load. The node's willingness to perform tasks can be regarded as a reputation assigned to the node, and 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:

$$\Psi(e,t) = crwr^{\alpha}(\Phi_{\mathcal{R}}(e), \Phi(t_{\mathcal{Q}})) \cdot \Phi_{ft}^{\beta}(e) \cdot \Phi_{fb}^{\gamma}(e) \cdot \lambda^{\delta}(e, t_{\mathcal{Q}}) \tag{3}$$

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

Exploration. Unlike the function `antChooseContact(·)` used by the colored ants, hunter ants combine all types of pheromone colors (Listing 2, line 8). However, the probability to choose link e' as the next hop is computed in a similar manner:

$$p_h(e',t) = \frac{e^{\frac{\Psi(e',t_{\mathcal{Q}})}{T_i}}}{\sum_{\forall e'' \in E_i} e^{\frac{\Psi(e'',t_{\mathcal{Q}})}{T_i}}} \tag{4}$$

TABLE II. NODE ATTRIBUTES

type	CPU freq.	cores	RAM	Nodes
Volunteer	$1-2$ GHz	$1-6$	$0.1-2$ GBs	$100-3,000$
Data Center	$1-3$ GHz	$2-32$	$2-6$ GBs	7

TABLE III. TASK ATTRIBUTES

type	duration	Cores	RAM	Deadline offset	Arrival mean
small	$0-0.4$ h	1	$0-0.5$ GBs	0.2	200 ms
large	$1-12$ h	$1-4$	$1-4$ GBs	0.4	600 ms

IV. SIMULATION

We evaluated our ACO-based instance of the framework using a volunteer cloud computing scenario (§IV-A), modeled in the discrete event simulation environment called DEUS [9], [10]. DEUS is a general-purpose, open-source, Java-based simulation environment, characterized by extreme ease of use and flexibility, which supports the analysis of complex and large scale systems. Our volunteer cloud simulator is realized as a DEUS project, and is available at http://bit.ly/18MunO4.

A. *Simulated Scenario*

In the following, we describe the main characteristics of the scenario used in the experiments. The network includes 10 cloud sites, among which 7 are managed by data centers and the others are purely P2P. The specification of the nodes' resources is reported in Table II. Volunteer nodes are less computationally powerful since they correspond to mobile devices such as laptops. 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 semi-hierarchical with supernodes which have connections with peers of other sites, and normal nodes which have connections only in the same site. Each node joining the network notifies its status (online, going offline) to the corresponding supernode, and receives a list of neighbors — a random subset of the volunteer nodes in the same site.

Each node acts both as task producer and consumer. Nodes share their resources to address tasks execution requests coming from other nodes, but can also create requests for their tasks. We consider that nodes execute tasks in exclusive application environments, allocating for this purpose a Virtual Machine (VM), which is released when the execution of the task is completed. A task is accepted for execution by a node only if the latter is able to guarantee its completion within its deadline, 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 models of underlying communication network described in [11].

The workload model we considered is the Google Cloud Backend [6], described in [12]. There, task requirements are characterized by CPU cycles and memory occupation. Since the workload data are partially obfuscated [6], we did some assumptions, such as a QoS (Quality of Service) parameter defined by a deadline (more restrictive for small tasks), after which task execution is considered to be useless. Tasks attributes are reported in Table III. The duration of the simulated scenario is of 1 hour, with a granularity of ten milliseconds.

The task arrival model is taken from [12], i.e., based on Markovian processes. The inter-arrival time between two consecutive tasks is modeled as an exponential random variable with mean value equal to 600 ms for large tasks, and 200 ms for small tasks. From a queue theoretic point of view, the scenario can be seen as a queue model where data centers are modeled as $M/G/m/\infty$ queues, while the volunteers are modeled as $M/G/1/\infty$ queues. I.e., task arrivals are modeled by a Markovian process (M), service time follows a generic (G) distribution, data centers have m VMs, volunteers have 1 VM each, and task queues are unbounded.

B. *Instantiated ACO Algorithm*

As described in §III, our ACO-based algorithm is highly parametric. The actual configuration used in the reported experiment has been specified into the XML configuration files of the DEUS tool (see Listing 3) shows the configuration corresponding to scout ants colored by finishing time. 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.).

In the experiments, the three resource scout ants are configured with: `ttl` $=3$, `initial pheromone` $=1$, `depositing function` $=x$, `aging function` $=1-x/5$, and `constant temperature` $=1$. The finishing time scout ants differs in the pheromone deposit function that must be decreasing (to assign more pheromone when the finishing time is closer to the actual time), 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 3 attempts for each task (hunting efforts before giving up), pheromone deposit function equals to $1-x$, the weight for each kind of pheromone (used in Eq. 3) is 1, and the constant temperature value equals to 1.

C. *Evaluated performance indicators*

The simulator allows to measure several performance indicators. Here we focus on those we consider particularly significant, to evaluate the goodness of our algorithm in terms of perceived QoS, communication overhead and fairness (load balance). In particular, we report the following indicators: (i) *Hit plus running rate*, which is the relative amount of tasks that meet their deadline or that, being still running, will likely complete if their host will not go offline; (ii) *Useless message rate*, which is the relative amount of refused requests over the total number of sent requests, indicating the overhead of the requests sent to overloaded nodes; (iii) *Mean task waiting time*: the time that a task spends in the queue, before its execution starts; (iv) *Mean task sojourn time*: the time that a task spends in the network, summing up waiting and execution time.

D. *Results*

Apart from the basic common configuration we described above, it is worth mentioning that every node uses ants configured with exactly the same behavior. We performed parametric simulations, to study the behavior of the system for

Listing 3. Colored scout ant configuration

```
<aut:event id="coloredAntFinishingTime" handler="it.imtlucca
    .aco.ColoredAntEvent" >
  <aut:params>
    <aut:param name="hasSameAssociatedNode" value="true" />
<! const = a, line = b * x + a, exp = a * e^{b*x} + c -->
    <aut:param name="antColor" value="finishingTime" />
    <aut:param name="initPheromone" value="1" />
    <aut:param name="ttl" value="3" />
    <aut:param name="pheromone_a" value="1" />
    <aut:param name="pheromone_b" value="-0.2" />
    <aut:param name="ant" value="line" />
    <aut:param name="evaporation" value="0.0001" />
    <aut:param name="pheromoneAging_a" value="1" />
    <aut:param name="pheromoneAging_b" value="-0.2" />
    <aut:param name="agingFunc" value="line" />
    <aut:param name="temperature_a" value="1" />
    <aut:param name="temperatureFunc" value="const" />
</aut:event>
```

different number of participating volunteer nodes and varying the frequency of scout ants release (from every 50 second to every $2,000$ seconds). In the following we refer to the average results obtained after reaching a 95% confidence interval, with a radius of 0.001, evaluated with the Student's t-test. Typically, tenths of simulations are needed, each taking several hours.

The purpose of the experiments was to evaluate the impact of scout ants, in the proposed algorithm. It is worth to remark that the algorithm can run without those ants by solely relying on the feedback pheromone collected by hunter ants. These experiments show that scout ants significantly improve the algorithm's performance in several dimensions.

Figures 2 (left), 3 (left) and 3 (right) report the Hit + Running Rate for all, large, and small tasks, respectively. The values are plotted considering the number of participating volunteer nodes on the horizontal axis. Obviously, the higher the number of nodes, the better the performance of the system is in terms of Hit + Running Rate. As expected if the scout ants are spawned with smaller frequency hunter ants have less information and thus make worse decisions. This happens especially when the number of nodes is higher and thus more scout ant explorations are required to build an informative computational field. The overall number of performed tasks is acceptable considering the limited number of participants.

In Fig. 2 (right) we report the Rate of Refused Requests for Remote Execution, which, as expected, is lower when the number of nodes increases. Scout ants allow a faster reduction in the refused requests thanks to the knowledge about resources availability introduced by them in the field.

Due to lack of space the rest of the performance indices results are only briefly commented. With a low number of nodes, the knowledge added by the scout ants and the mismatch policy followed in Eq. 2 tends to favor large tasks to data center nodes, leading to an increased waiting and sojourn times, with 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. Scout ants increase the number of accepted remote requests. With them, the load is better spread among the nodes which are able to execute the tasks. The only drawback is perceived in the increased waiting and sojourn times, due to the bottleneck created by the large tasks.

V. Related Work

The ACO approach was firstly proposed by Di Caro and Dorigo [13], 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. Our algorithm is inspired by this work, but addresses the more complex problem of distributed QoS-constrained task execution.

Another source of inspiration for our work is the comprehensive survey on approaches to network routing and load-balancing based on ACO, by Sim *et al.* [14]. The authors stress the main weakness of ACO-based approaches, namely *stagnation*, and focus 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 consider *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* (a privileged set of ants may release more pheromone than the rest) and *pheromone-heuristic control* (the choice of ants is a weighted combination of the amount of pheromone and the estimate of a heuristic). Such techniques can be easily implemented and evaluated in our framework.

Some authors have tried to adopt existing ACO-based approaches to solve load balancing problems in task distribution systems [15]–[17]. Many of them apply the basic *minmax* algorithm proposed by Di Caro and Dorigo [13]. Unfortunately, such works do not describe their algorithms in sufficient detail to allow us to implement and evaluate them in our framework. Nevertheless, we discuss some of their main concepts.

Mishra [15] proposed a simple ACO approach to deal with the load balancing problem intended as the fact that every node does 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 that plays a role similar to 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 makes a random decision. An extended version of this 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, such an approach is suitable for load balancing in network routing problems, not for collaborative task execution in volunteer clouds, as ants' decisions do not take into account the QoS requirements of the tasks.

LBACO (Load Balancing Colony Optimization) [16] is an extension of the basic ACO algorithm of [13]. LBACO not only tries to find the optimal resource allocation for each task, but also 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

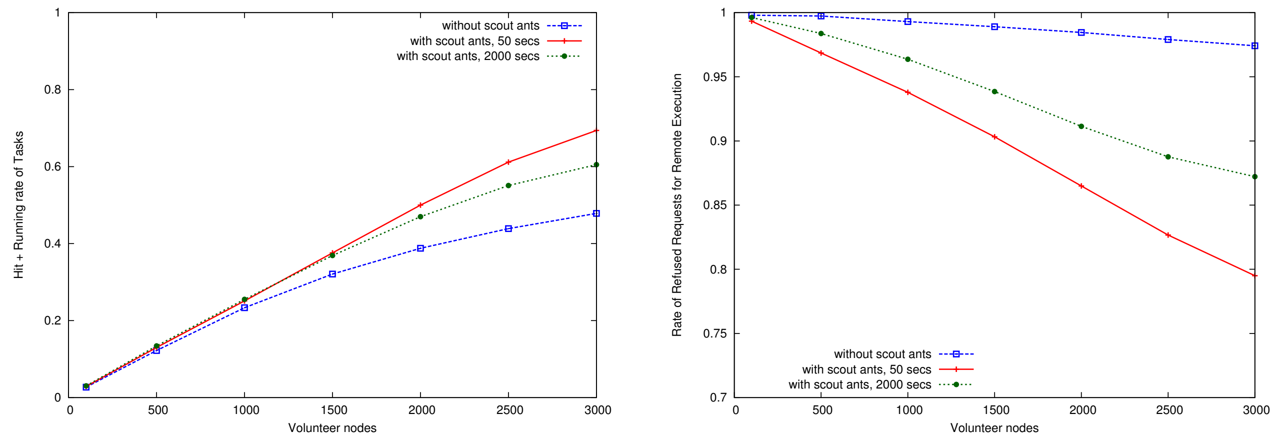

Fig. 2. Hit + Running Rate (left), and Rate of Refused Requests for Remote Execution (right) of the tasks, with and without scout ants.

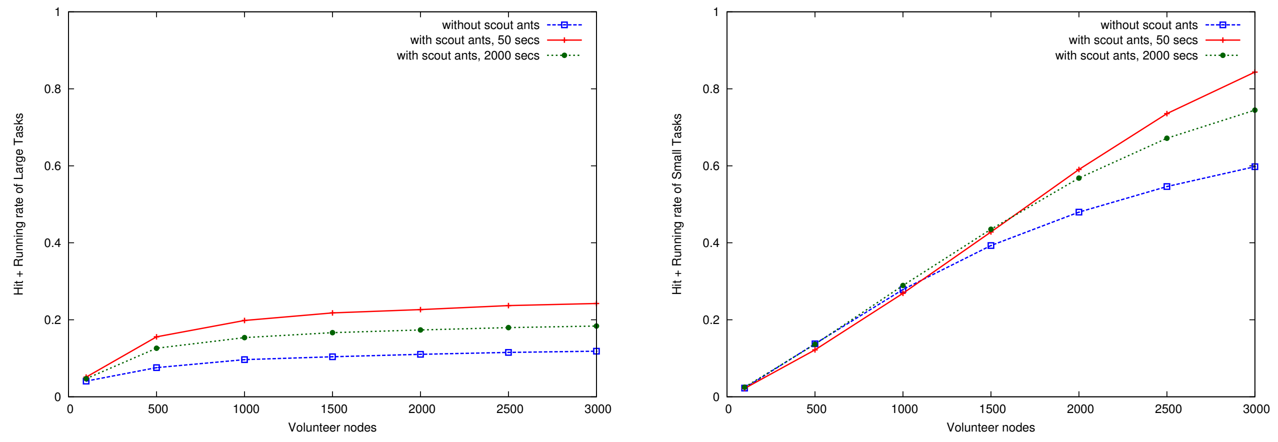

Fig. 3. Hit + Running Rate of large (left), and small (right) tasks, with and without scout ants.

difference among the task that completes first, and the one that complete last, in a task set. The basic ACO algorithm is extended by carrying out new task scheduling, depending on the results in the past scheduling, and also considering the load of each VM. The algorithm takes into account VM characteristics like: the number of processors available in each VM, its MIPS (Million Instruction Per Second) capability and the communication bandwidth. The LBACO algorithm is evaluated through simulation, comparing it with basic FIFO and ACO algorithms, in terms of the average makespan and the Degree of Imbalance (a measure of imbalance among VMs). Our work has a different purpose: it considers only individual tasks, that have an associated deadline parameter, and tries to maximize the number of tasks completed respecting their QoS requirements. The LBACO cannot be directly applied for collaborative task exception in volunteer clouds, since it assumes that each node knows all the resources available in the neighbors nodes, which is unrealistic in those scenarios.

The idea of colored ants was previously presented in a completely different way by Ali and Belal [17]. They considered a multiple colony approach, where each node sends a colored colony throughout the network. Using colored ant colony helps in preventing ants of the same nest from following the same route, and hence enforcing them to be distributed all over the nodes in the network. One main difference with respect to our work is that their ants tend to maximize the coverage of the network (exploration), while the strategy of our scout ants can be configured with a certain exploration-exploitation tradeoff, according to the softmax method (see Eq. 1).

Our hunter ants share many similarities with the *spatial computing* paradigm [4]. The use of decentralized approaches for managing Grid resources in a P2P fashion through a spatial computing approach was first tackled by Di Stefano and Santoro [18], where a job resource request is defined by a capsule, which is characterized by mass and energy, and moves on a three-dimensional surface. The surface is built on top of the overlay network (where the nodes define the X-Y plane) and the available node's resource characterize their mass (adding the Z dimension). The capsule moves according to a couple of functions, which define the difference of potential among neighbors nodes (i.e., the capsule's behavior, according to its remaining energy), and the friction (which causes a loss of energy of the capsule, thus ensuring termination). Such an approach associates one surface to each type of resource. In our ACO algorithm, hunter ants follow an approach which can be considered an extension of the one proposed by Di Stefano and Santoro [18]. Each scout ant releasing its colored pheromone contributes to the construction of a surface where the values are not associated to the node itself but to the link. Moreover task requests do not have their own mass, but specify how they react over different surfaces. Hunter ants are able to combine these colored surfaces, to build a new "normalized surface" (Eq. 2) to the task requests and to the importance of each kind of resource (through the weights η_k). In our algorithm the Z dimension is given by the under/over utilization of resources, since our approach tries to minimize the amount of resources reserved and not used by the task. This surface normalization process aims to combine the different surfaces generated by the pheromone colors, and at same time it is able to take into account one of the ants' goals (the minimization of task wasted resources). Hunter ants behave similarly to task capsules since their next hop choice is guided by this surface. The links that prove to be more attractive to the combined colored pheromone will present a higher gradient, guiding the hunter towards it. Differently to a traditional spatial computing approach, hunter ants do not have their own energy that must be exhausted to stop the exploration, but adopt a more clever approach, stopping when they find a suitable node that can fulfill their requests. This ensures to find a solution in less time, which is more effective when coping with scenarios where tasks may have stringent deadlines. Termination is ensured by the ant's TTL.

Finally, we remark that we use our cloud simulator instead of CloudSim [19] (a popular simulator for cloud computing environments) since CloudSim imposes a rigid architecture that is not suitable for volunteer clouds. More precisely, cloud agents in CloudSim must submit a description of their capabilities to a broker that receives the task execution requests and then dispatch them. These centralized solution does not copy well with the de-centralized nature of volunteer clouds.

VI. Conclusions and Future Work

We presented two novel contributions in the field of volunteer cloud computing. First, a flexible framework for the design and evaluation of agent-based algorithms for collaborative cloud computing problems. The key feature of the framework is a shared data structure called Computational Colored Field, inspired by Ant Colony Optimization [3] and Spatial Computing [4]. Overall, the framework is also inspired by the *volunteer computing* [1] and *cloud using agents* paradigms [2]. The proposed general framework can be easily instantiated in different ways, to better fit the characteristics of the considered scenario.

Second, inspired by previous ACO and spatial computing based approaches to distributed computing problems (e.g., [13]–[18]), we presented an instance of the framework in the form of a novel, highly parametric ACO-based algorithm, which we advocate as a strong candidate for collaborative task execution problems. The proposed ACO approach is self-adaptive, which makes it suitable for dynamic scenarios such as volunteer clouds, where nodes can join and leave the network at any time. The benefits of the algorithm can be summarized by its decentralized and self-* nature together with a light network overhead introduced by ants. The proposed algorithm was evaluated with a set of simulation-based experiments using workload data from Google [6], [12].

We plan to evaluate further features of our algorithm, with particular attention to the novel anti-stagnation mechanisms we have proposed (angry ants and memory aging). Moreover, we plan to investigate novel mechanisms based on heterogeneous ants (i.e., ants having different behaviors), as well as standard spatial computing approaches based on local information diffusion rules. We shall also consider existing volunteer computing platforms such as the ScienceCloud [20].

Acknowledgments

Research partly supported by the European projects IP 257414 ASCENS and STReP 600708 QUANTICOL, and the Italian PRIN 2010LHT4KM CINA.

References

[1] V. D. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa, "Volunteer computing and desktop cloud: The cloud@home paradigm," in *IEEE International Symposium on Network Computing and Applications*, july 2009, pp. 134–139.

[2] D. Talia, "Cloud computing and software agents: Towards cloud intelligent services," in *WOA'11*, vol. 741. CEUR, 2011, pp. 2–6.

[3] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[4] F. Zambonelli and M. Mamei, "Spatial computing: An emerging paradigm for autonomic computing and communication," in *WAC'04*, ser. LNCS, vol. 3457. Springer, 2004, pp. 44–57.

[5] M. Amoretti, A. Lluch-Lafuente, and S. Sebastio, "A cooperative approach for distributed task execution in autonomic clouds," in *PDP*. IEEE Computer Society, 2013, pp. 274–281.

[6] J. L. Hellerstein, "Google cluster data," Google research blog, Jan. 2010, http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[7] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[8] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ser. IMC '06. ACM, 2006, pp. 189–202.

[9] M. Amoretti, M. Agosti, and F. Zanichelli, "DEUS: a discrete event universal simulator," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques for Communications, Networks and Systems, (SimuTools 2009).* ICST, 2009, p. 58.

[10] "Distributed Systems Group, DEUS," http://code.google.com/p/deus/.

[11] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup," in *6th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS '13).* ICST, 2013.

[12] A. Mishra, J. Hellerstein, W. Cirne, and C. Das, "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[13] G. D. Caro and M. Dorigo, "Antnet: A mobile agents approach to adaptive routing," IRIDIA, Tech. Rep., 1997.

[14] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 33, no. 5, pp. 560–572, 2003.

[15] R. Mishra and A. Jaiswal, "Ant colony optimization: A solution of load balancing in cloud," *International Journal of Web & Semantic Technology (IJWesT)*, vol. 3, no. 2, pp. 33–50, April 2012.

[16] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," in *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*, aug. 2011, pp. 3–9.

[17] A.-D. Ali and M. A. Belal, "Multiple ant colonies optimization for load balancing in distributed systems," in *Proceedings of ICTA 2007*, 2007.

[18] A. Di Stefano and C. Santoro, "A peer-to-peer decentralized strategy for resource management in computational grids: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 9, pp. 1271–1286, 2007.

[19] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[20] "The science cloud platform," http://svn.pst.ifi.lmu.de/trac/scp/.