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
University of 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 cloud technologies offers new opportunities for distributed and collaborative computing. Volunteer clouds are a prominent example, where participants join and leave the platform and collaborate by sharing computational resources. The high complexity, dynamism and unpredictability of such scenarios call for decentralized self-* approaches. We present in this paper a framework for the design and evaluation of self-adaptive collaborative task execution strategies in volunteer clouds. As a byproduct, we propose a novel strategy based on the Ant Colony Optimization paradigm, that we validate through simulation-based statistical analysis over Google workload data.

Categories and Subject Descriptors

C.1.4 [**Parallel Architectures**]: Distributed architectures; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Distributed Algorithms, Distributed Architectures, Performance

Keywords

cloud computing, volunteer computing, self-* systems, ant colony optimization, bio-inspired algorithms, spatial computing, peer-to-peer, distributed tasks execution

1. INTRODUCTION

The wide adoption of the cloud computing paradigm is increasing the efforts of the research community on investigating and developing approaches and techniques for engineering cloud-based platforms. A special attention is being devoted to resource management and optimization. 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* [15] 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).

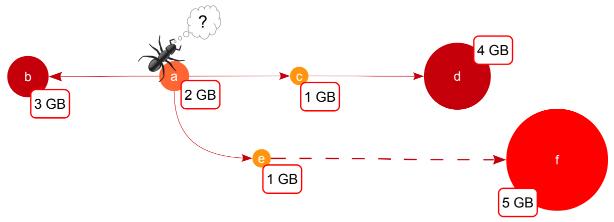


Figure 1: Where should the ant go?

The success of the volunteer paradigm is witnessed by a wide set of existing platforms—among others, we mention: BOINC [8], HTCondor [33], OurGrid [11], Seattle [13] and Seti@home [9]. Most of them rely on DHTs (Distributed Hash Tables) to save and locate resources in the network. Task execution requests are submitted (often in batch mode) to a centralized *node manager*, in charge of finding the most promising nodes to execute them. Such a centralized control mechanism is a weakness, because of the central point of failure problem, and may also be a bottleneck when the number of the involved volunteer nodes scales to the size of a typical cloud. Moreover, maintaining a global knowledge of the load on each node is clearly hard to be realized in practice.

As global coordination and optimization techniques can be hardly applied to volunteer clouds, the attention has shifted to the adoption of agent-based techniques, as advocated in [32], like Ant Colony Optimization (ACO) [19] and Spatial Computing [34]. Such approaches provide flexible and scalable solutions to distributed computing problems, such as collaborative task execution (see e.g. [17] and the references therein).

In this paper, we present a framework for the design and engineering of highly efficient volunteer clouds with collaborative task execution. Such a framework is as an evolution of our preliminary work [6], which we extend here in several directions. First, in §2, we introduce 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, in §3 we define a highly parametric ACO-based algorithm, offering a decentralized solution characterized by lightweight ant agents, which maintain and exploit the colored computational field without requiribng any additional data structures. In §4, we validate our approach by reporting an excerpt of the experimental evaluation of the ACO-based algorithm we have conducted, where we assess the performance of various alternatives and parameters, and we compare the algorithm against some standard solutions. Our benchmark is based on the workload described by the Google Cluster dataset [20]. In §5, we discuss our main sources of inspiration and further related work. Finally, in §6 we provide some concluding remarks and outline our current and future research efforts. [1]

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.

2. 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 anytime, and submit and satisfy task execution requests, subject to QoS requirements. When a node is not able to execute a task of its own, it needs to find another "collaborator" node able to do it. Such a search must take into account that, as in social networks and the Internet's autonomous systems, 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.

Figure 1 illustrates a situation in which a node a creates an (hungry ant) agent, to search for a collaborator in a network. The depicted network includes nodes b, c, d, e and f, some of their connections (as arrows) and some additional nodes (not depicted for simplicity). The dashed edge from node e to node f denotes an arbitrarily long path between both nodes. Even if the picture includes the information about memory resources within nodes (which is also emphasized by the color intensity and node size), node a and its agent are totally unaware of such information. Thus, a blind strategy such as random walk seems the only option for the agent. Even in presence of information, it is not clear which node should be selected: node f is the one offering more memory, but may lie too far away; node a is the direct neighbor with the highest memory capacity; node d is not a direct neighbor but can be reached in two hops and has higher memory capacity than b.

The approach we propose offers several parametrizable solutions to these problems. The basic idea is to create a supporting structure, to help agents in taking their decisions. Figure 2 illustrates the situation described above, on a network whose links have been enriched with information related to the amount of resources that may be found by following them. We call such a supporting structure the *Colored Computational Field*. In the particular example of the figure, the field corresponds to the memory resource only, where edge labels denote values related to the amount of memory. For a more appealing visualization, such values are also emphasized with the edge color intensity and thickness. The concrete field being depicted assigns to the link between a and c the highest rank, reflecting the fact that a near node with high memory capacity (d) can be found through it. As we shall see, our algorithms will use such information to make probabilistic choices in the quest for collaborators.

DEFINITION 2.1 (COLORED COMPUTATIONAL FIELD). *Let K be a set of $\mathbb{R}^+$-valued computational pheromones. A* K-colored

[1]An extended version of this work is available online for the convenience of the reviewers. It includes an appendix with some additional results and a snapshot of the colored computational field for the memory, the cpu core and their combined version.

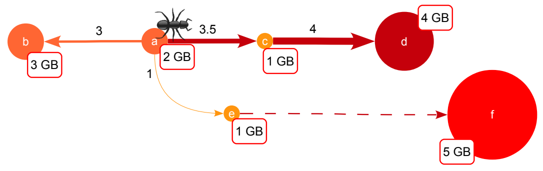


Figure 2: Memory field supporting the ant's quest.

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 $\Phi : E \to \mathbb{R}^{|K|}$ 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* (i.e., an estimate of when a node will complete its current tasks) and a *feedback pheromone* (i.e., a rate based on how often the node accepts and successfully executes tasks) as elements of K in our examples. For the sake of simplicity we assume that all nodes feature the same set of resources, which are measurable in $\mathbb{R}^+$ (i.e., as non-negative reals).

The subset of QoS requirements of a task t, to be interpreted as lower bounds on computational resources $\mathcal{R}$, is denoted by a vector t_Q in $(\mathbb{R}^+)^{|\mathcal{R}|}$. 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 pheromone values in $\mathbb{R}$. 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, the i-th element of a vector $\vec{x}$ is denoted as $\vec{x}_i$. Sometimes, we 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* [34], used to ease resource discovery, while minimizing communication.

3. ACO-BASED ALGORITHM

Several algorithms can be defined on top of a Colored Computational Field, including those based on local diffusion rules, which are typical of spatial computing approaches [34]. This section presents a novel paradigmatic example of a highly parametric ACO-based algorithm. The algorithm uses ant-like mobile agents that are in charge of maintaing and exploiting the Colored Computational Field. It relies on two different types of ants: *colored scout ants* and *hunter ants*. Colored scout ants (Listing 1 lines 1-2) periodically explore the neighborhood of a node, to discover computational resources and 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 (Listing 1 lines 4-8) 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.

Listing 1: Ant Main Cycle

```
1  ∀ color $k \epsilon \mathcal{K}$ with $period_k$:
2    $ant_k$.coloredAntStep(sourceNode);
3
4  when $n \epsilon N$ generates a task $t$:
5    if ($n$ cannot execute $t$):
6      while (an executing node $n'$ is not found) &&
7          (∃ hunterAnt attempts):
8        $ant$.coloredAntStep($n'$);
```

Listing 2: Colored scout ant algorithm

```
1  $ant_k$.coloredAntStep(Node $n$){
2    $ant_k$.pathAdd(n);
3    $ant_k$.saveNodeGoodness($n$, $k$);
4    $ant_k$.updateTtl();
5
6    if ($ant_k$.getTtl()>0){
7      next := choose with probability
8                            $p_k((\mathrm{n}, next), E_n \setminus \{ant_k.path()\})$;
9      if (nextNode != null){
10       $ant_k$.coloredAntStep(next);
11       return;
12     }
13   }
14   previousNode := $ant_k$.getPreviousNode();
15   $ant_k$.coloredAntStepBack(previousNode, node);
16 }
17
18 $ant_k$.coloredAntStepBack(Node to, Node from){
19   new_$\phi_k$ := $ant_k$.getMemoryAgingPheromone(to, from);
20   if ( (new_$\phi_k$ > $\phi_k(to, from)$) || (k != FINISHING_TIME ))
21       $\phi_k(to, from)$ := new_$\phi_k$;
22   previousNode := $ant_k$.getPreviousNode();
23   $ant_k$.coloredAntStepBack(previousNode, to);
24 }
25
26 $ant_k$.getMemoryAgingPheromone(Node to, Node from){
27   memoryTrace := $\vec{\phi}$(subPath(to, from));
28   best_$\phi_k$ := max(memoryTrace);
29   return agingDiscount(pathLength(best_$\phi_k$, to));
30 }
```

The two types of ants are described in detail in §3.1 and §3.2, respectively. It is worth to remark a common key feature: they both exploit the field to make 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 inverse proportion to the distance to resources) and *angry ants* (a third kind of agents that remove pheromones along outdated links).

3.1 Colored Scout Ants

Colored scout ants are periodically spawned in a process that is independent from the request and execution of tasks (Listing 1 lines 1-2). 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 up to hunter ants. Listing 2 describes the behavior of scout ants by means of pseudocode. Each scout 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 14), releasing the pheromone according to a memory aging approach (line 26). Below, we provide a detailed explanation of the main features of the algorithm.

Choosing the Next Hop: Temperature-dependent Exploration & Exploitation. The behavior of ants is based on online Reinforcement Learning (RL) [27], where at each step the decision of which link to explore next involves a choice among *exploration* (try to gather new information) and *exploitation* (focus 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 [27]. Each ant selects its next hope by taking into account both the past path desirability (exploitation) and to the exploration compliance, according to the rate established by the following equation

$$\pi_k(e_j) = e^{\frac{\Phi_k(e_j)}{T_i}} \tag{1}$$

The probability $p_k(e_j, E')$ that the k-colored scout ant at node i chooses $e_j \in E' \subseteq E_j$ is then defined by

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

Typically, E' is selected to contain all the 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 tradeoff, i.e., if $T_i \to \infty$ the ant at node i tends to follow a more random approach (all paths have the same preference), while if $T_i \to 0$ the ant follows a greedy approach, which reduces the exploration component. If instead, if T is close to 1, the choice tends to be proportional to the pheromone of each link. For instance, with $T = 1$, in the situation depicted in Figure 2, a memory scout ant having to take a decision on a will choose among the links towards nodes b, c and e with probability 0.4, 0.47 and 0.13, respectively.

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 re-initialized to encourage the exploration of new resources.

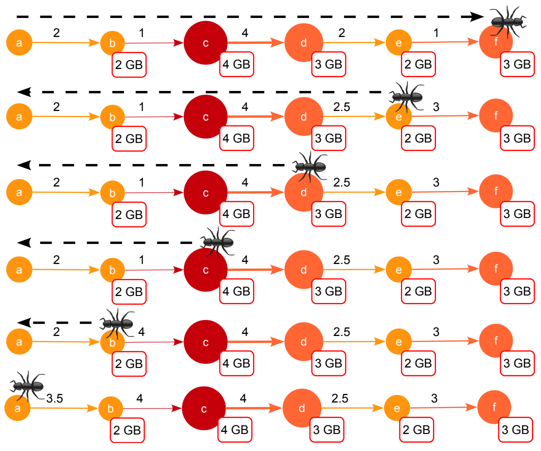


Figure 3: Scout ants depositing pheromone with memory aging.

Discounting the Distance: 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, depending on the ant memory aging factor (to prevent stagnation) and the node goodness in that part of the network (Listing 2, lines 26-30, $\texttt{getMemoryAgingPheromone}(\cdot)$).

We do not use the traditional concept of aging, where ants deposit lesser and lesser pheromone as they move 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 but also for scout ants spawned at other nodes.

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 from the best resource found so far, and not 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 $\texttt{agingDiscount}(\texttt{mem\_aging on best\_}\phi)$, illustrated in Listing 2 at line 29, which, for example, could be instantiated as $\texttt{best\_}\phi - \texttt{mem\_aging} \cdot \texttt{AgingFactor}$, where $\texttt{best\_}\phi$ is the best value found so far, $\texttt{mem\_aging}$ is the distance from it and the discounting factor is $\texttt{AgingFactor}$

Fig. 3 exemplifies the memory aging approach, where the aging discouting function is instantiated in the same way as the previous example, and $\texttt{AgingFactor} = 0.5$. The figure represents a part of the field that has been already explored by some scout ants, which have been spawned at different nodes (not depicted for simplicity). The scout ant is spawned at node a and follows the path $a \rightarrow b \rightarrow c \rightarrow e \rightarrow e \rightarrow f$ (Fig. 3, top). When the TTL expires, after 5 hops (at node f), the scout ant returns home (node a). In the first step back, the actual value of the resource at node f (i.e., 3) is taken into account (see the label on the link from node e to d). Note, however, that in the second step back the pheromone is updated 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, as in the third step back (from c to b), where the ant has found a resource with value 3, but the previous pheromone is 4 (which

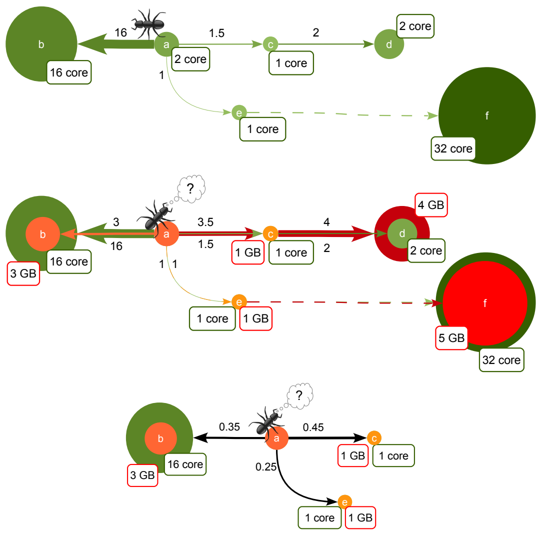


Figure 4: CPU field (top), overlap of memory and CPU fields (center) and their combination with the allocation heuristic.

may be the result from a previous exploration of some other nodes, not depicted in the figure).

Discounting Information Age: 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 previously released one. Instead, if pheromone values would be updated regardless of the best previously found values, they would be highly variable, thus providing unstable information. We consider resources such as the amount of RAM and CPU characteristics to be non-volatile, as they cannot be allocated forever but only on a 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, it uses angry ants (described below) to update the field.

Dealing with Dynamic Networks: Angry Ants. Despite the non-volatile nature of resources, the unstable nature of the network of participants [31] 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.

3.2 Hunter Ants

When a node has a task for which it cannot respect the deadline, it starts spawning multiple *hunter ants* (Listing 1 lines 4-8). Every hunter ant tries to find a node ready to satisfy the task execution request. To this purpose, the hunter ant starts exploring the network, by exploiting 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 requirement —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), following the Colored Computational Field (line 8) built according to the overall pheromone—see the explanation of Eq. 5 below. 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.

Combining and Minimizing Resources: Resource Allocation Heuristic. When choosing the next hop, hunter ants need to take into account the various pheromones of the field, which may offer contradictory information. As a simple example, consider the memory field of Figure 2 and the CPU field on top of Figure 4. By overlapping them, as in the center picture of Figure 4, we illustrate the contradictory information: the memory-colored pheromone (red) tends to promote the link from a to c (since it leads towards node d, which has a good memory capability), while the CPU-colored pheromone (green) tends to promote the link from a to b (which is a node with lots of cores). Our approach is based on a weighted sum of both fields, as we shall see.

An additional issue is the global need to maximize the number of tasks that meet their deadline. Such a problem is clearly intractable 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 completely used) will increase the probability to accommodate more requests in the future. These heuristics rely on two functions, namely the *single resource waste ratio srwr* and the *combined resource waste ratio crwr*, defined in Eq. 3 and 4, respectively. Note that the latter function uses a vector η of size $|\vec{x}|$, allowing to express preferences among resources.

$$srwr(x, y) = \frac{min(x, y)}{max(x, y)} \tag{3}$$

$$crwr(\vec{x}, \vec{y}) = \sum_{\forall k \in 1..|\vec{x}|} \frac{\eta_k \cdot srwr(\vec{x}_k, \vec{y}_k)}{\sum_{\forall \sigma \in 1..|\vec{x}|} \eta_\sigma} \tag{4}$$

Such functions are exemplified in Table 1, 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 required resources $\vec{x}$ perfectly match the provided ones $\vec{y}$. The other cases exemplify the mismatches due to under/over resource utilization. Smaller values of $crwr$ suggest higher mismatch degree between requested and provided resources.

A concrete example is illustrated at the bottom of Figure 4, where memory and CPU pheromones are combined by a hunter ant, looking for a node able to execute a task that requires 2 GB of memory and 2 cores. The labels of the black edges outgoing from node a are the result of evaluating $crwr(\langle 2, 2\rangle, \Phi(e'))$ for $e' \in \{(a, b), (a, c), (a, e)\}$ and with all η being identical.

It is worth to note that, even if the required resources are above the ones that seem to be provided by a link, following it may still

Listing 3: Hunter ant algorithm

```
ant.antStep(Node n){                                              1
  ant.pathAdd(n);                                                 2
  ant.updateTtl();                                                3
  if (askExecutionToNode(node, t)){                               4
    previousNode := ant.previousNode();                           5
    ant.antStepBack(previousNode, n);                             6
  } else if (ant.getTtl()>0) {                                    7
    nextNode := antChooseContact($p_k(E_n \setminus \{ant.path()\})$, t);  8
    if (nextNode != null){                                        9
      ant.antStep(nextNode);                                      10
      return;                                                     11
    }                                                             12
  }                                                               13
                                                                  14
  ant.antStepBackHome(ant);                                       15
}                                                                 16
                                                                  17
ant.antStepBack(Node to, Node from){                              18
  ant.depositPheromone(to, from);                                 19
  previousNode := ant.getPreviousNode();                          20
  ant.antStepBack(previousNode, to);                              21
}                                                                 22
                                                                  23
depositPheromone(Ant ant, Node from){                             24
  new_$\phi_{fb}$ := ant.agingPheromone(to);                       25
  $\phi_{fb}(e_{from})$ = new_$\phi_{fb}$;                          26
}                                                                 27
```

Table 1: 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

be worth. Indeed, the link may lead to a region of the network with nodes having with the required resources. This is the reason why the pheromone of all the three links in Figure 4 is considered by the hunter ant's decision mechanism.

Weighting links. Such heuristic functions are used to associate goodness values to links. To 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 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, 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:

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

Table 2: 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 3: 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

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 which 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. 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 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 rate

$$\pi_h(e', t) = e^{\frac{\Psi(e', t_Q)}{T_i}} \tag{6}$$

4. VALIDATION

We evaluated our ACO-based framework in a volunteer cloud computing scenario modeled[2] in the discrete event simulation environment DEUS [7, 16]. 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. Simulation experiments have been enriched with the functionalities offered by MultiVeStA [26,29], a distributed statistical analysis tool. MultiVeStA provides a language (MultiQuaTEx) to express the system properties of interest in a compact fashion. Such properties are evaluated by performing independent distributed simulation runs, until the required accuracy is met.

4.1 Simulated Scenario

In the following, we describe the main characteristics of the scenario used in the experiments. The network of participants includes 10 cloud sites, of which 7 sites have a central data center and several volunteer nodes, while the rest of the sites are composed by volunteer nodes only. The specification of the node resources is reported in Table 2. Volunteer nodes are less computationally powerful, as 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 connected with peers of other sites, and normal nodes which have connections within the same site only. Each node joining the network notifies its presence to the corresponding supernode, and receives a list of neighbors — a random subset of the volunteer nodes in the same site.

[2]Source code is available at `http://bit.ly/18MunO4`.

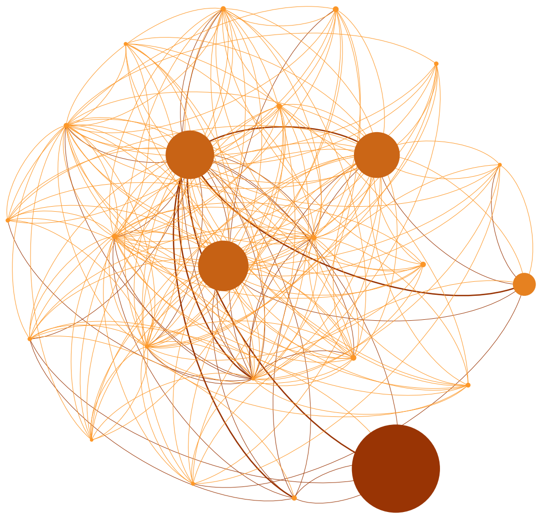


Figure 5: Pheromone field for a network with 30 nodes.

Nodes are both task producers and consumers. Nodes share their resources to address tasks execution requests coming from other nodes, but can also create requests for their tasks. Tasks are executed in exclusive application environments. A task is accepted for execution only if its timely completion can be guaranteed, 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. [28].

The workload model we considered is the Google Cloud Backend [20], described by Mishra et al. [24]. There, task requirements are characterized by CPU cycles and memory occupation. The tasks attributes we have considered are reported in Table 3, namely task duration, required number of cores, required RAM, deadline offset and arrival mean. As workload data are partially obfuscated [20], we had to make some assumptions. An example is the quality of service, in terms of task deadlines, after which task executions are considered to be useless. We have considered a deadline offset with respect to the actual duration of the task.

The arrival mean determines the task arrival process, which is Markovian, as derived by Mishra et al. [24] from the Google Cloud Backend traces. 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. The duration of the simulated scenario is 1 hour, with 10 ms granularity.

4.2 Instantiated ACO Algorithm

As described in §3, our ACO-based algorithm is highly parametric. The actual configuration of the algorithm can be specified in XML configuration files of the DEUS tool (see Listing 4). 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.).

The common parameters of the scout ants used in all experiments are specified in Listing 4. The configuration of the pheromone deposit function depends on the color. For computational resources (CPU frequency, CPU cores and memory) the function is defined as x, while for color "finishing time" the function must be decreasing (to assign more pheromone when the finishing time is closer to the actual time), thus it is configured with $1 - x/5$.

Hunter ants are instead configured with 3 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 Eq. 5) equal to 1, and a constant temperature value of 1.

An example of the pheromone released in the network, corresponding to the memory resource color, is sketched in Figure 5. For the sake of clarity, the depicted network contains only few of the thousands of nodes. The graph follows the convention for which the edge direction is codified by the curved arrow, in a clockwise way from the source to the target node [21]. It is possible to notice that the pheromone suggests the ants to go towards the data center node with the largest memory capacity. Such diagrams as a useful tool to intuitively visualize the evolution of the colored field and to detect anomalies or interesting phenomena like unexpected over/under-accumulation of pheromones in some network regions.

Listing 4: Colored scout ant configuration

```
<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="evaporation" value="0.0001" />
<aut:param name="pheromoneAging_a" value="1" />
<aut:param name="pheromoneAging_b" value="-0.2" />
<aut:param name="agingFunc" value="1-x/5" />
<aut:param name="temperature_a" value="1" />
<aut:param name="temperatureFunc" value="1" />
```

4.3 Evaluated performance indicators

Our simulator allows to measure several performance indicators. Here we focus on those we consider particularly significant for the evaluation of our algorithm in terms of perceived QoS, communication overhead and fairness (load balance). In particular, we report the following indicators: (i) Hit 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 task execution 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 of a node, before its execution starts; (iv) Mean Task Sojourn Time: the time that a task spends in the network, summing up waiting and execution time.

4.4 Results

Apart from the basic common configuration we described above, it is worth mentioning that every node uses ants that are configured with exactly the same behavior. We performed parametric simulations, to study the behavior of the system for different number of participating volunteer nodes and varying the frequency of scout ants release (from every 50 second to every 2000 seconds).

The proposed ACO-based algorithm is compared with other approaches, namely random walk, round robin and *greedy oracle*. When a node uses the random walk approach, execution requests are randomly spread to the neighborhood of the node that has generated the task, not considering its requirements neither the node resources. The round robin approach probes the neighbors nodes in a circular order. To better compare the performance with the ACO algorithm, which uses a limited amount of ants with a pre-specified TTL, also these two algorithms have a TTL and maximum number of requests they can send. It is worth to note that the random walk approach can be considered as an instance of our framework, with $T \to \infty$ and without scout ants. The *greedy oracle* is an algorithm that is impossible to realize in practice, as it assumes to have complete information (knowing all nodes' resources and the queued tasks) and assigns the task to the node that can complete the new task for first. Thus, from local optimal choices, it 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 a near-to-optimal performance.

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. 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 took 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. However, the results show that scout ants significantly improve the algorithm's performance in several dimensions.

Figures 6 (bottom left), 6 (top left) and 6 (top right) report the Hit Rate values for all, large, and small tasks, respectively. Such 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 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, i.e. 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.

In Fig. 6 (right) we report the Useless Message Rate, which, as expected, is lower when the number of nodes increases. The Greedy Algorithm is not included as, having global knowledge, it always sends a request to a node that will accept it . In general, the round robin and random approaches perform better than our ACO algorithm, but augmenting the frequency of scout ant release leads the ACO algorithm close to the performance of the random approach. Indeed, frequent scout ants allow a faster reduction of refused requests, thanks to the knowledge about resources availability they introduce in the field.

Due to lack of space, we do not show the results related to the other performance indices. Nevertheless, we summarize the insights we obtained. With a low number of nodes, the knowledge added by the scout ants and the mismatch policy followed in Eq. 4 tends to favor large tasks to data center nodes, leading to 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 and 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 in the increased waiting and sojourn times, due to the bottleneck created by large tasks.

All in all, our algorithm ACO with frequent scout ant release offers the best performance in terms of perceived QoS at a reasonable price in terms of communication overhead.

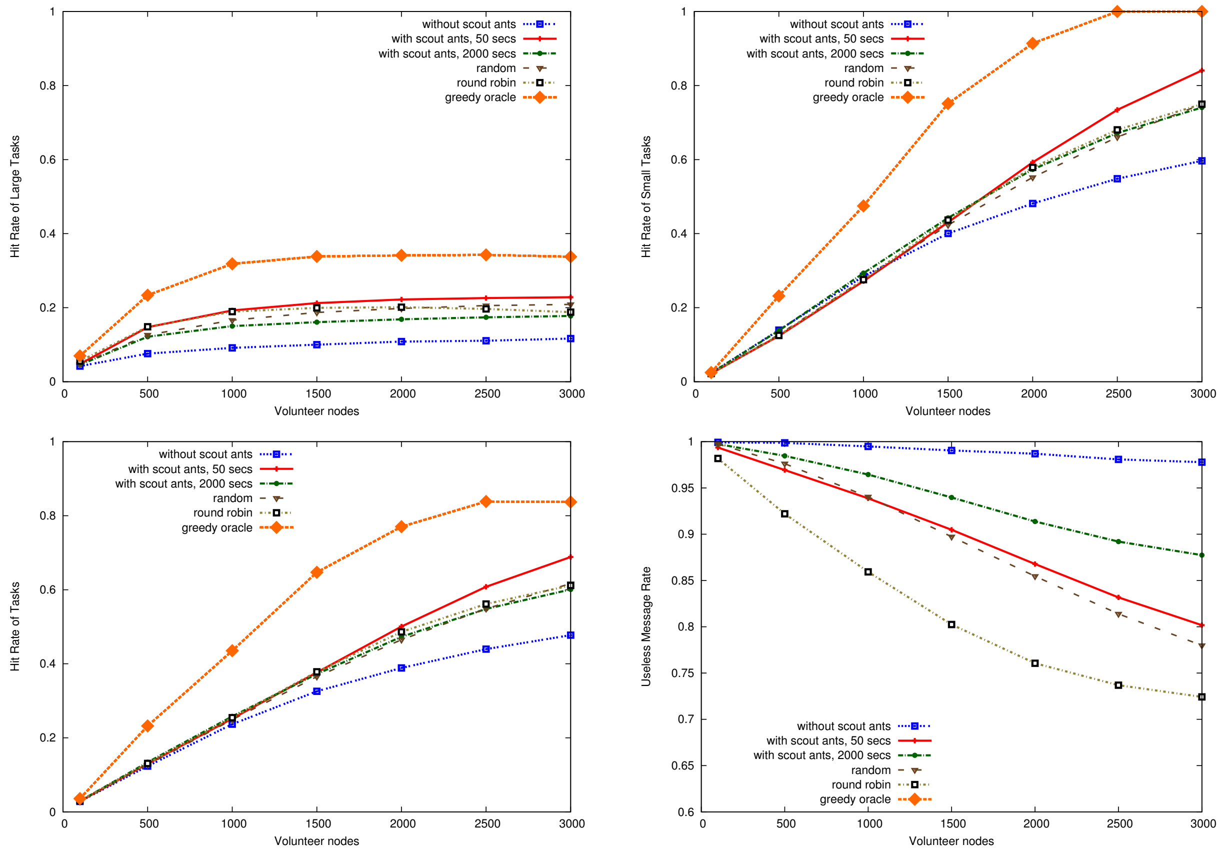


Figure 6: Hit Rate for large (top left), small (top right) and all (bottom left) tasks, and Useless Message Rate (bottom right).

5. RELATED WORK

ACO-based approaches. The ACO approach was firstly proposed by Di Caro and Dorigo [14], 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 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 *et al.* [30] 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 adopted ACO-based approaches to solve load balancing problems in task distribution systems [5, 23, 25]. Many of them apply the *minmax* algorithm of Di Caro and Dorigo [14]. Unfortunately, such works do not describe their algorithms in sufficient detail to allow their implement and evaluation in our framework. Nevertheless, we discuss some of their main concepts.

Mishra [25] 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, 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.

LBACO (Load Balancing Colony Optimization) [23] is an extension of the basic ACO algorithm of [14]. 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 complete last. The basic ACO algorithm is extended by carrying out scheduling decisions that take into account the results of previous scheduling decisions, and also considering the load of each VM. The algorithm takes into account VM characteristics like the number of processors available at each VM, its MIPS (Million Instruction Per Second) capability and communication bandwidth. The LBACO algorithm is evaluated through simulation, and compared with basic FIFO and ACO algorithms, in terms of average makespan and Degree of Imbalance (a measure of imbalance among VMs). Our work 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. The LBACO cannot be directly applied to collaborative task execution 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 [5]. They considered a multiple colony approach, where each node sends a colored colony throughout the network. Using colored ant colonies helps 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), while our scout ants can be configured with a certain exploration-exploitation tradeoff, according to the *softmax* method (see Eq. 2).

A different approach was adopted by Di Nitto *et al.* [17], who used bio-inspired algorithms to balance the workload, i.e., to ensure that all the nodes have almost the same amount of tasks in their queue. To deal with node heterogeneity, the algorithm proceeds in two steps: in the first one, the network is rewritten to cluster the nodes of the same type; in the second step, messages are spread among nodes of the same type, to redistribute the load. The goal of such an algorithm is different from our one's. Moreover, we consider nodes with heterogeneous resource characteristics, not belonging to only few classes.

Spatial Computing Approaches. Our hunter ants share many similarities with the *spatial computing* paradigm [34]. 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). One surface is associated 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, by 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. 4) depending on the specific task request 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. Such a 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, as their next hop choice is guided by the surface. The links that are more attractive to the combined colored pheromone will present higher gradients, guiding the hunter towards them. Differently to a traditional spatial computing approach, hunter ants do not have their own energy that must be exhausted to stop the exploration. Instead, they adopt a more clever approach, stopping when they find a suitable node, which can fulfill their requests. Such an approach allows 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.

Cloud Simulators. Finally, we remark that we use our cloud simulator instead of CloudSim [12] (a popular simulator for cloud computing environments), because CloudSim imposes a rigid architecture, which is not suitable for volunteer clouds. More precisely, cloud agents in CloudSim must submit a description of their capabilities to a broker, which receives task execution requests and dispatches them. Such a centralized solution does not cope with the de-centralized nature of volunteer clouds.

6. 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 [19] and Spatial Computing [34]. Overall, the framework is also inspired by the *volunteer computing* [15] and *cloud using agents* paradigms [32]. 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., [5, 14, 18, 23, 25, 30]), 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 [20, 24].

We plan to evaluate further features of our algorithm, with particular attention to the self-* anti-stagnation mechanisms we propose here (i.e. 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 [10]. Finally, time will be devoted to evaluate the performance of our algorithm with different grid workload traces and node traffic models available on the web such as: [1–4, 22].

Acknowledgments

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

7. REFERENCES

[1] The grid workloads archive. http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php.

[2] The internet traffic archive. http://ita.ee.lbl.gov/.

[3] Logs of real parallel workloads from production systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[4] Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/.

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

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

[7] M. Amoretti, M. Picone, F. Zanichelli, and G. Ferrari. Simulating Mobile and Distributed Systems with DEUS and ns-3. In *Proc. Int.'l Conference on High Performance Computing and Simulation (HPCS)*, Helsinki, Finland, July 2013.

[8] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[9] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, Nov. 2002.

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

[11] F. Brasileiro, E. Araiijo, W. Voorsluys, M. Oliveira, and F. Figueiredo. Bridging the high performance computing gap: the ourgrid experience. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 817–822, 2007.

[12] 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.*, 41(1):23–50, Jan. 2011.

[13] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. In S. F. et al., editor, *SIGCSE*, pages 111–115. ACM, 2009.

[14] G. D. Caro and M. Dorigo. Antnet: A mobile agents approach to adaptive routing. Technical report, IRIDIA, 1997.

[15] 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*, pages 134–139, july 2009.

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

[17] E. Di Nitto, D. Dubois, and R. Mirandola. On exploiting decentralized bio-inspired self-organization algorithms to develop real systems. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS '09. ICSE Workshop on*, pages 68–75, 2009.

[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.*, 19(9):1271–1286, 2007.

[19] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evolutionary Computation*, 1(1):53–66, 1997.

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

[21] D. Holten and J. J. van Wijk. A user study on visualizing directed edges in graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2299–2308. ACM, 2009.

[22] W. Kim, A. Roopakalu, K. Y. Li, and V. S. Pai. Understanding and characterizing planetlab resource usage for federated network testbeds. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 515–532. ACM, 2011.

[23] 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*, pages 3–9, aug. 2011.

[24] A. Mishra, J. Hellerstein, W. Cirne, and C. Das. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters. *ACM SIGMETRICS Performance Evaluation Review*, 37(4):34–41, 2010.

[25] R. Mishra and A. Jaiswal. Ant colony optimization: A solution of load balancing in cloud. *International Journal of Web & Semantic Technology (IJWesT)*, 3(2):33–50, 2012.

[26] MultiVeStA website. http://code.google.com/p/multivesta/.

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

[28] 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.

[29] S. Sebastio and A. Vandin. MultiVeStA: Statistical model checking for discrete event simulators. In *VALUETOOLS*, 2013.

[30] 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*, 33(5):560–572, 2003.

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

[32] D. Talia. Cloud computing and software agents: Towards cloud intelligent services. In *WOA'11*, volume 741, pages 2–6. CEUR, 2011.

[33] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[34] F. Zambonelli and M. Mamei. Spatial computing: An emerging paradigm for autonomic computing and communication. In *WAC'04*, volume 3457 of *LNCS*, pages 44–57. Springer, 2004.

APPENDIX

This appendix contains additional material for the interested reader. In particular, §A describes additional performance parameters, §B presents the corresponding measures in the evaluated scenario and §C includes some pictures of colored fields for a small network.

A. PERFORMANCE PARAMETERS

The following is an exhaustive and detailed description of the performance parameters that we take into account in our experiments and that are actually logged in our statistical analysis simulations:

1. **Hit Rate**: defined as the relative amount of tasks that have completed (satisfying their deadline) or are still running, over the overall number of sent requests. A running tasks will certainly be completed if the node that have taken it in charge does not go offline. A higher value is desirable since it denote the ability of a strategy to accommodate a heavier load.
2. **Useless Message Rate** (*Refused Rate*): defined as the relative amount of refused requests over the total number of sent requests. Such a performance indicator enables us to evaluate the overhead introduced by sending requests to overloaded nodes. A request is repeatedly sent until a node able to accept it is found, or the nodes list is exhausted. It is preferable to have low values of this parameter corresponding to a minor overhead introduced in the network, due to useless messages.
3. **Total Number of Addressed Execution Requests** (*Tot req*): defined as the total amount of requests spread in the network in order to find a node able to execute the task. Similarly to the previous parameter, but in absolute values, this one allows to point out the introduced overhead.
4. **Mean Task Waiting Time** ($\overline{W}$ *time*): defined as the time that a task spends before that its execution starts. Lower values suggest a better system response to the incoming task requests.
5. **Mean Task Sojourn Time** ($\overline{S}$ *time*): defined as the overall time that a task spends in the network. It is measured as Waiting plus Execution times, i.e., it combines the previous performance indicator with the required service time, thus also in this case lower values are better.
6. **Mean Number of Tasks per Node** ($\overline{tpn}$): the mean number of tasks that have been executed by each node taking part in the network. The more this value is closer to $\frac{\text{numberOfTasks}}{\text{numberOfNodes}}$ and the more the strategy follows a load balancing approach, i.e., spreads the workload evenly among the nodes.
7. **Task Variance per Node** (σ^2_{tpn}): the variance of tasks executed by each node taking part in the network. A reduced variance suggests that more nodes execute an amount of work close to the mean.
8. **Max Number of Tasks per Node** ($\max(tpn)$): the number of tasks executed by the node that has worked most. It is an index of the effort done by the node that have executed more tasks.
9. **Min Number of Tasks per Node** ($\min(tpn)$): the number of tasks executed by the node that has worked less. A higher value suggests that all the nodes have an ”active“ participation to the network executing tasks.

B. ADDITIONAL EXPERIMENTAL DATA

Figures 7–12 present additional experimental data of the experiments presented in §4.

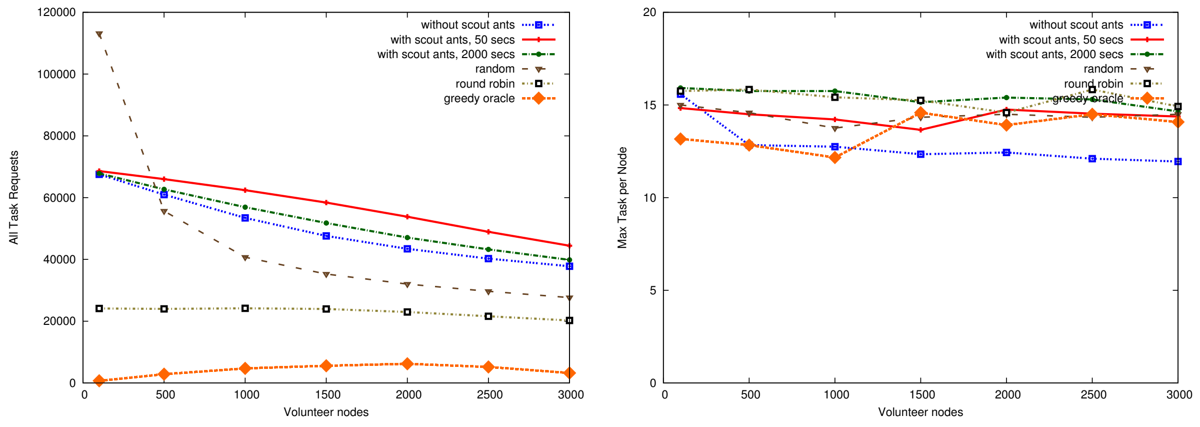


Figure 7: All remote execution requests (left) and Max number of tasks executed per node (right)

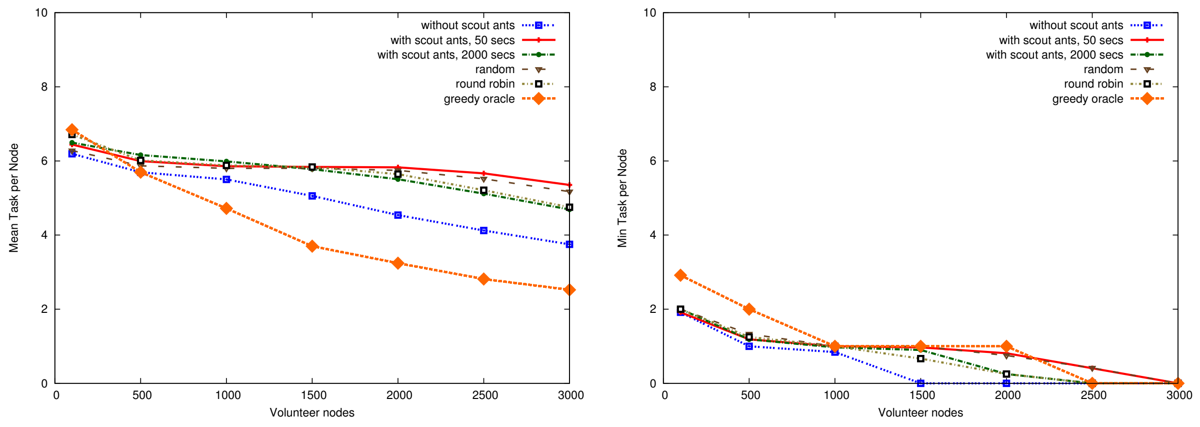


Figure 8: Mean (left) and Min number of tasks executed per node (right)

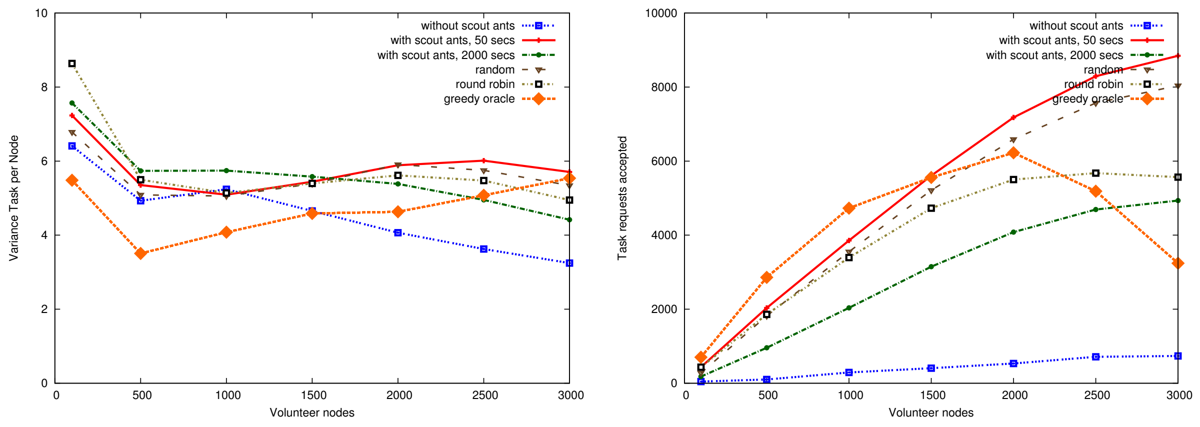
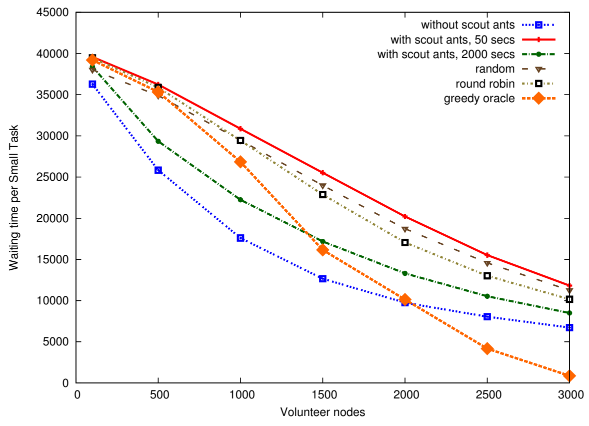


Figure 9: Variance of tasks executed per node (left) and accepted remote execution requests (right)



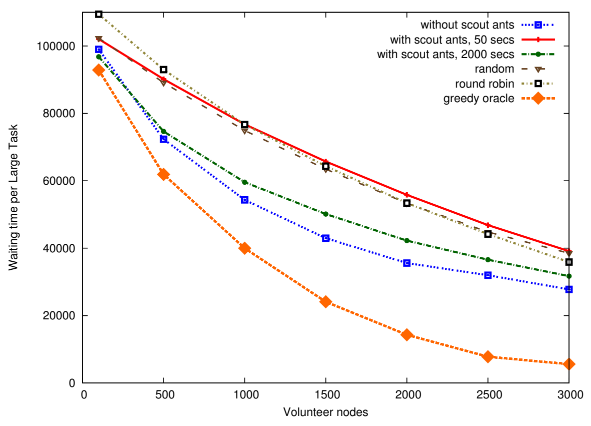
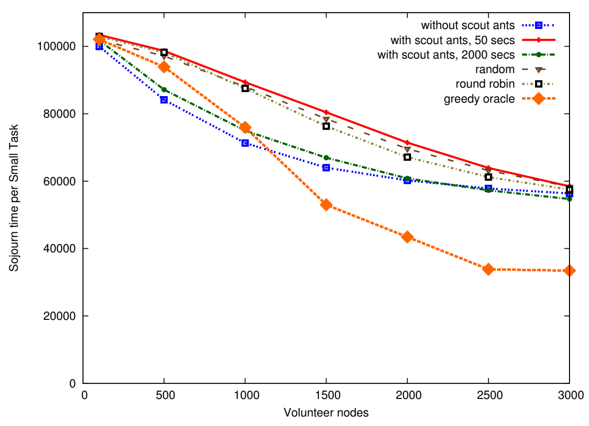


Figure 10: Mean Waiting time for small (left) and large (right) task



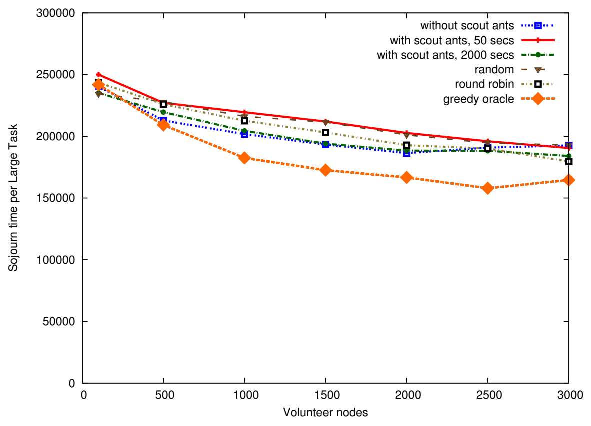
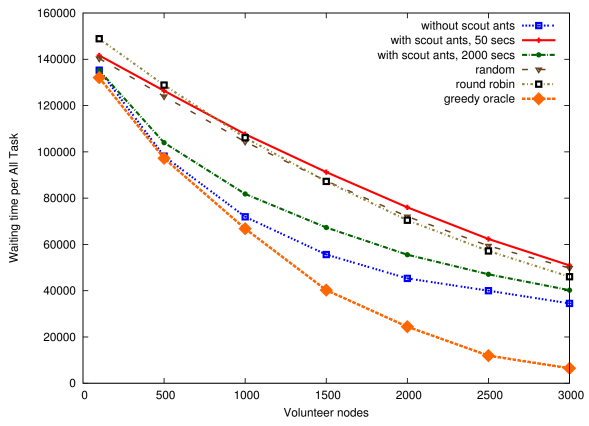


Figure 11: Mean Sojourn time for small (left) and large (right) task



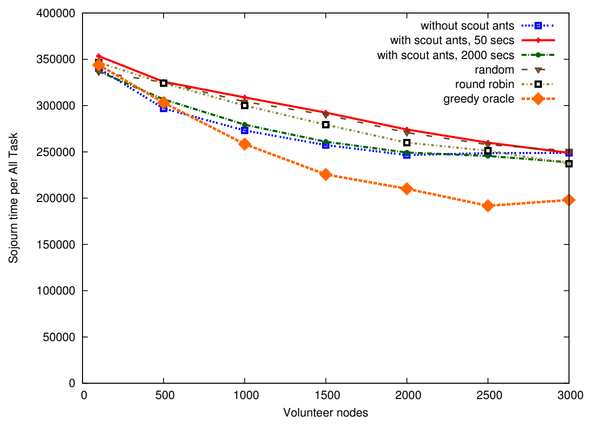


Figure 12: Mean Waiting (left) and Sojourn (right) time for all type of task

C. COLORED COMPUTATIONAL FIELDS

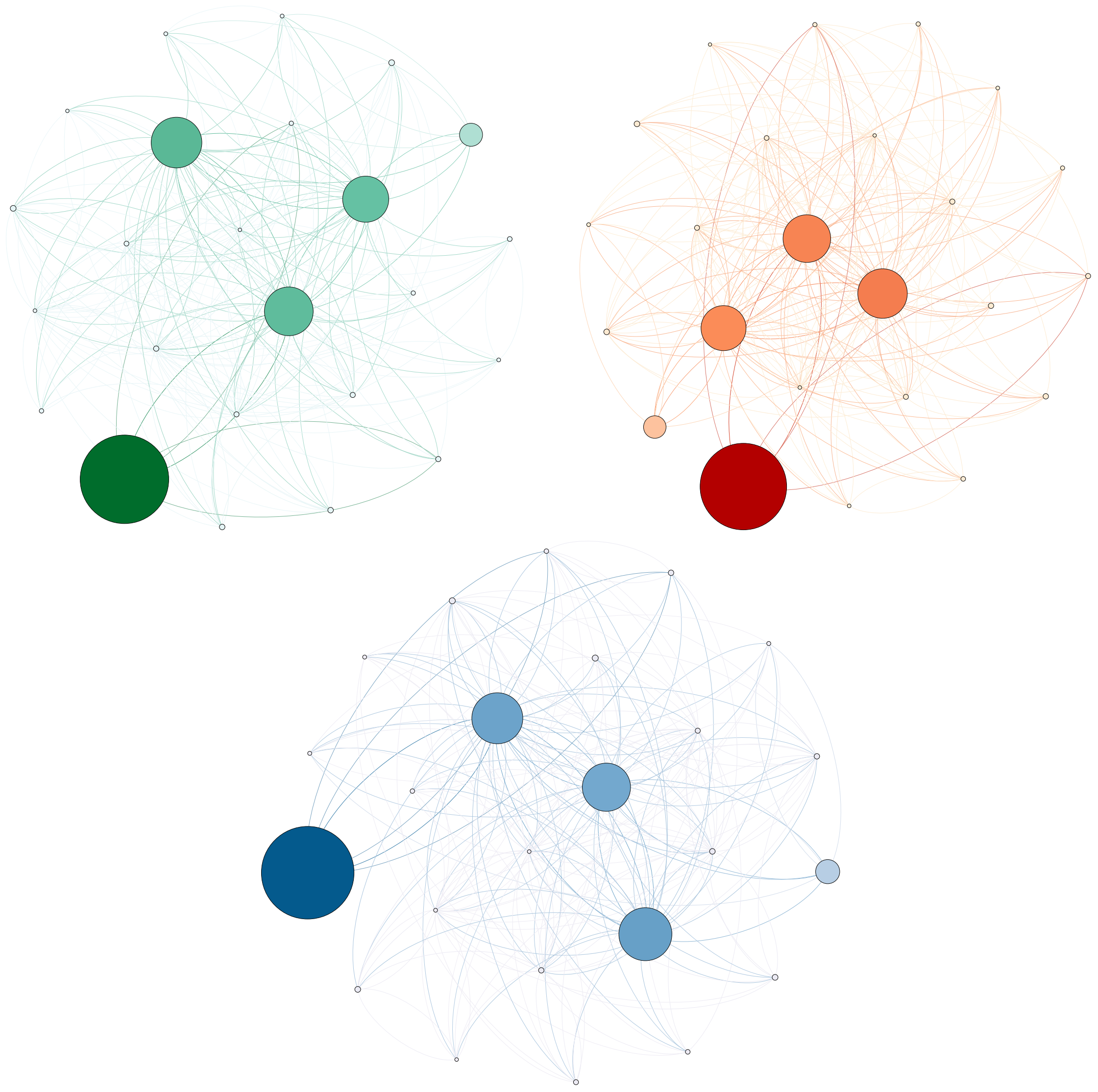


Figure 13: Colored fields: CPU core (top left) and Memory (top right) and combined (bottom)