Reputation-based Cooperation in the Clouds *

Alessandro Celestini[1], Alberto Lluch Lafuente[2], Philip Mayer[3], Stefano Sebastio[2], and Francesco Tiezzi[2]

[1] Istituto per le Applicazioni del Calcolo, IAC-CNR, Rome, Italy
a.celestini@iac.cnr.it

[2] IMT Institute for Advanced Studies, Lucca, Italy
{a.lluch,s.sebastio,f.tiezzi}@imtlucca.it

[3] Ludwig-Maximilians-Universität München, Germany
mayer@pst.ifi.lmu.de

Abstract. The popularity of the cloud computing paradigm is opening new opportunities for collaborative computing. In this paper we tackle a fundamental problem in open-ended cloud-based distributed computing platforms, i.e., the quest for potential collaborators. We assume that cloud participants are willing to share their computational resources for shared distributed computing problems, but they are not willing to disclosure the details of their resources. Lacking such information, we advocate to rely on reputation scores obtained by evaluating the interactions among participants. More specifically, we propose a methodology to assess, at design time, the impact of different (reputation-based) collaborator selection strategies on the system performance. The evaluation is performed through statistical analysis on a volunteer cloud simulator.

Keywords: Autonomic Computing, Cloud Computing, Reputation, Trust

1 Introduction

Cloud computing has gained huge popularity in recent years. This is mainly due to the progress in virtualization technologies and the transfer of data centers to low-cost locations. This emergent paradigm meets many of today's requirements like the need of elaborating big volumes of data or the necessity of executing applications of which only the front-end is able to run on a mobile device. Next to the presence of traditional cloud computing platforms built running in proprietary data centers, another trend that is gaining popularity is the use of volunteer resources offered by institutions or ordinary people for, e.g., scientific computations. The success of the volunteer paradigm is witnessed by the wide set of existing platforms where, amongst others, we mention: BOINC [3], HTCondor [24], OurGrid [8], Seattle [9] and Seti@home [4]. These collaborative environments can effectively be seen as cloud computing platforms where

* Research partially supported by the EU through the FP7-ICT Integrated Project 257414 ASCEns, STReP project 600708 QUANTICOL, and the Italian PRIN 2010LHT4KM CINA.

participants take advantage of virtualization techniques to share their computational resources for distributed computing applications, like the execution of tasks. Differently from grid computing, we cannot expect volunteer participants to guarantee a certain level of performance in terms of shared resources or online availability. On the other hand, volunteer clouds offer the unique opportunity of letting participants find their collaborators in the entire volunteer network. The quest for collaborators is one of the key aspects in such platforms.

The contribution of the paper is twofold: (1) a reputation-based approach to the collaborator selection problem, and (2) a methodology to assess, at design time, the impact of the selection strategies on the system performance. We focus on a peer-to-peer cooperative environment on top of which a cloud platform offers a task execution service. The aim of the platform is to maximize the number of successfully executed tasks. We consider a cloud platform with an integrated reputation system, where a reputation score is associated to each node denoting the trustworthiness of the node. Reputation scores are computed on the basis of the rating values released by other nodes. These ratings evaluate the behavior of the node in past interactions. Specifically, we exploit the concept of reputation as an indicator of the likelihood that a node will successfully execute the task, i.e., the higher the reputation the higher the probability that the task will be successfully executed. We assume that tasks have an associated Quality of Service (QoS) requirement given by a deadline, after which the task execution is considered unsatisfactory.

The reputation-based node selection strategies provide *loose coupling* and *self-adaptivity*, since the nodes take their decisions based on the reputation learning mechanism. Overall, the system is able to autonomously adapt the load of nodes during system execution while avoiding to interact with nodes to check their current status. This is in particular useful in platforms that are *dynamic*, where nodes can join and leave the system continuously over time, and *heterogeneous*, since participants with different computational resources are rated with the same mechanism but can customize their strategies according to their needs.

As a reference case study for experimenting with the proposed reputation-based approach, we have used the Science Cloud [16,6] platform, developed within the EU project ASCENS [5], which aims at offering resources for scientific computations. In particular, we have modeled this cloud platform with DEUS [2] and carried out a number of experiments considering different configuration scenarios. The obtained results show the benefit of the use of reputation-based approaches. Specifically, our experimental analysis shows that a probabilistic reputation-based strategy (compared to both reputation based and random approaches) seems to be more robust to the workload variation, offering the best performance at a reasonable communication overhead.

Structure of the paper. Section 2 introduces the case study. Section 3 presents the node selection strategies. Section 4 presents a simulation-based comparison of the strategies' performances on different instances of the case study. Section 5 discusses related work. Finally, Section 6 closes the paper and suggests directions for future work.

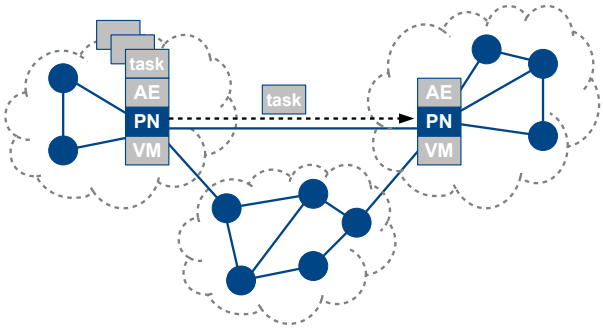

Fig. 1. Science Cloud Architecture

2 Science Cloud Case Study

The Science Cloud [16,6] is a volunteer P2P Platform-as-a-Service (PaaS) system developed within the European project ASCENS [5] with the aim of creating a decentralized platform for sharing computational resources in scientific communities. Participants contribute with their desktops, mobile devices, servers, or virtual machines by running platform nodes on them. Since we assume that every participant runs exactly one platform node, we will often use *node* and *participant* as synonyms. Nodes may be *heterogeneous*, i.e., they may offer different virtual resources (CPU, disk, memory) and also highly *dynamic*, i.e., they may enter or leave the system at any time, and their load as well as their resources may change. Self-healing network mechanisms take care of such dynamism to ensure that all nodes are able to route messages between themselves. The Science Cloud provides distributed application execution as its main functionality. Applications may range from batch tasks to more sophisticated human-interactive applications, which may have different requirements in terms of resources (e.g., CPU and memory needed) and QoS (e.g., deadlines).

Fig. 1 illustrates the general architecture of the Science Cloud. For the sake of simplicity, we provide the internal architecture of two internal nodes only. The platform nodes (PN) run on top of virtual machines (VM) offering an application environment (AE) where tasks can be executed. For each task, one *initiator* node is chosen as being responsible for processing the task (not necessarily executing the task itself). This node needs to be secure against failures, i.e., if it goes down another node needs to take its place. The initiator may choose to execute the task itself, but may also choose to delegate to a collaborator node. Whoever finally executes the task is called the *executor*.

In this work, we consider scenarios based on batch task applications. We assume that a deadline is associated with each task and that a task is successfully executed if the deadline is met. Moreover, each task requires only one executor node. Since the Science Cloud is a cooperative environment, we consider scenarios without malicious nodes: nodes accept a task only if they satisfy the resource requirements of the task and if they estimate that they are able to ex-

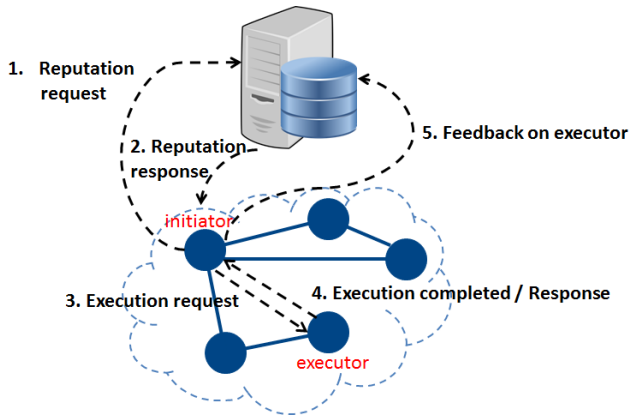

Fig. 2. Finding a collaborator

ecute it. These estimations assume that the node will remain always connected. However, nodes are not aware of their online/offline times and thus, it may happen that a node accepts a task but, before finishing its execution, goes offline. When a node goes offline it loses all the tasks in its queue regardless of the time in which it will return online. An offline node that returns online will maintain its identifier; this is indispensable to describe the node behavior.

Our goal in such scenarios is to maximize the overall number of tasks executed. To achieve this goal we aim at selecting, for each new task, the node most likely to successfully execute the task. The next section proposes some reputation-based strategies to address this.

3 Node Selection

In this section we briefly introduce reputation systems and, in particular, we recall the definition of the Beta [14] reputation system. Then, we suggest and discuss some reputation-based strategies based on the Beta reputation system for addressing the node selection problem. Each strategy consists of a *node ranking schema* and a *node selection strategy*.

First, we briefly describe (see Fig. 2) the underlying architecture and the protocol that nodes follow to implement the reputation-based strategies: (1) the initiator node sends a request to the reputation system asking for a list of potential executor nodes ordered according to the node selection strategy; (2) the reputation system provides the desired answer to the initiator node; (3) the initiator node starts contacting the potential executor nodes using the obtained list; (4) the contacted nodes send their response to the initiator, either rejecting the request or accepting it (and eventually communicating the completion of the task execution); (5) the initiator node provides feedback to rate its interaction with the contacted executor nodes. Note that the reputation system is not necessarily centralized. The use of a P2P overlay infrastructure enables the

use of distributed storage techniques [15], for example Distributed Hash Tables (DHTs), to store reputation values without a central coordinator.

3.1 Reputation Systems

A reputation system associates a reputation score to each node, denoting the trustworthiness of the node, i.e., the higher the reputation, the more trustworthy the node. The reputation score of each node is computed on the basis of the rating values released by other nodes. Such ratings correspond to evaluations of the behavior of the nodes in past interactions, which in our case can have only two possible outcomes: 'satisfactory' (i.e., the task was executed and its QoS was satisfied) or 'unsatisfactory' (the task was not executed or its QoS was not satisfied). In other words, we consider *binary* ratings. In our approach, the reputation of a node is an indicator of the probability that the node will successfully execute the task. We consider the termination deadline as the QoS parameter and we assume that missing a deadline makes the task completion useless.

In this work, we focus on *probabilistic trust* systems [12,13] which use probability distributions to model the behavior of a node. The goal of such systems is to provide an estimation of the distribution's parameters modeling the node behavior on the basis of past interaction outcomes, i.e., the ratings. This estimation is indeed the reputation score of the node and is used to compute the probability of future interaction outcomes with it.

For the definition of our strategies we exploit the Beta reputation system [14]. The name of this system is due to the use of the Beta distribution to estimate the posterior probabilities of binary events. In the Beta system, the behavior of each node is modeled as a Bernoulli distribution with success probability $\theta \in [0, 1]$. This means that, when interacting with a party whose behavior is (determined by) a given θ, the estimated probability that a next interaction will be satisfactory is θ. The reputation computed by the system is then an estimation $\tilde{\theta}$ of the node's behavior θ. Specifically, to compute the reputation of a given node, the Beta reputation system takes as input the number α of past satisfactory interactions with the node and the number β of past unsatisfactory interactions. The reputation $\tilde{\theta}$ of the node is given by the expected value of a random variable ϑ distributed according to the Beta distribution $Beta(\alpha + 1, \beta + 1)$, with $\alpha \geq 0$, $\beta \geq 0$, that is defined as follows:

$$\tilde{\theta} = \mathrm{E}[\vartheta] = \frac{\alpha + 1}{\alpha + \beta + 2}$$

Summing up, in our case the reputation of a node denotes the likelihood that the node, if selected, will not disconnect before completing the task and that it will accept the task because it is not overloaded, i.e., it can meet the deadline. Thus, nodes with high reputation should be able to successfully execute a task with higher probability.

	accept	**reject**	**complete**	**fail**
score		–	+	–

Table 1. Node ranking schema

3.2 Node Ranking Schema

The interactions we aim at evaluating in our systems are the following:

- **accept**: the selected node accepts the task;
- **reject**: the selected node rejects the task, since it cannot meet the deadline;
- **complete**: the selected node successfully completes the task execution, i.e., it meets the deadline and does not go offline during the execution;
- **fail**: the selected node fails in executing the task because it goes offline during the execution.

Notably, we assume that the executor nodes are truthful: they are able to accurately predict the task completion time and accept a new task *iff* they are principally able to execute it within the task deadline. However, nodes do not know their online-offline cycles a priori. It is thus possible that a node misses a task it has accepted by going offline.

Each action can be evaluated by the nodes as satisfactory (+) or unsatisfactory (–), which corresponds to giving a positive or negative rating, respectively. The node ranking schema is defined by the value assigned to each individual interaction as summarized in Table 1. Notably, the negative rating assigned to the action **fail** is given for each task whose execution was not successful. Notice also that no rating is given in case of task acceptance.

3.3 Node Selection Strategies

Reputation scores are used by the initiator node for the selection of an executor. We consider the following node selection strategies:

- **Random (`R`)**: a node is chosen randomly using an uniform probability distribution over the node pool (i.e., reputation is not taken into account).
- **Reputation-based (`RB`)**: the node with the highest reputation score is chosen. If more than one node exists with the same score, the choice is arbitrary (i.e., random).
- **Probabilistic reputation-based (`PRB`)**: a node is chosen randomly using a probability distribution over the node pool. Such a distribution assigns a probability to each node that is proportional to the reputation score of the node, i.e., the higher the node reputation, the higher the probability the node is selected. The idea is to introduce some randomness to avoid congesting nodes with good reputation, and also some fairness by giving nodes with low scores the chance to achieve a higher ranking (again). The probability that a given node i will be selected among l nodes (the node-i's neighbors) according this strategy is defined as follows:

$$P(select_i) = \frac{\tilde{\theta}_i}{\sum_{j=1}^{l} \tilde{\theta}_j}$$

4 Validation

In this section, we present an experimental validation of the proposed approach. We start by presenting our volunteer cloud computing simulator built on top of DEUS (Section 4.1). Then, we describe the performance parameters taken into account during our experimental validation (Section 4.2). We describe next the studied scenarios (Section 4.3) and conclude with reporting and discussing the obtained results (Section 4.4).

4.1 The simulator in a nutshell

Our simulation model is implemented in DEUS [2], an open-source discrete event simulation tool developed in Java. Although many other simulation tools are available, DEUS appears to be more generic and flexible regarding the analysis of complex dynamic systems like the one considered in this paper. The DEUS API allows users to define and characterize three type of components: (i) *nodes*, basic elements that interact in the complex system; (ii) *events*, which are the actions performed by the simulator and correspond to internal and external node actions that can be due to interaction among nodes; (iii) *processes*, which constrain the timeliness of events in a stochastic or deterministic manner. The node state transition can be defined by functions specified in the node source code or in the source code of the events associated to the node.

Our statistical analysis has been performed with MultiVeStA [19,22], a distributed statistical analysis tool that can be integrated with any discrete event simulator. MultiVeStA provides a language (MultiQuaTEx) to express the system properties of interest in a compact fashion. Essentially, MultiVeStA performs independent distributed DEUS simulation runs until these properties are evaluated with the required accuracy.

The simulator implements the basic machinery to suitably model the scenarios under consideration. In the following we discuss some parameters of the configurations of the simulator that can be taken into account to set up the desired volunteer cloud scenarios.

Workload Model. Tasks are generated by initiators according to some parametric process that determines frequency of task generation, their duration (expressed as CPU cycles) and memory occupation. Tasks are defined by their duration and their deadline. If the deadline expires the task execution is considered to be useless. The deadline offset is defined as 20% beyond the ideal task duration. Thus, if a task that requires `t_exec` arrives at time `t_arrival`, the task execution is considered useful if it is completed within time: `t_arrival + t_exec + task_exec*`20%.

Network Model. When a node accepts a task execution request coming from another node a communication overhead is evaluated to the simple yet realistic network models described by Saino *et al.* [20].

Node Model. The nodes realize an exclusive task execution environment where the whole Virtual Machine (VM) is assigned to only one task at a time. Its behavior is modeled by a $M/G/1/+\infty$ queue using the Kendall's notation [7], i.e., Poisson arrival process, general service time distribution with only one VM and infinity queue capacity. A task is accepted by a node only if the node is able to satisfy the requested task deadline, taking into account the tasks already on its queue but without knowing its departure time (i.e., the point in time when it goes offline). Thus, it is possible that a node accepts a task since it is able to satisfy the QoS constraint, but after a while it leaves the network losing the task execution results until that point. In this case the task is lost.

Node Classes. There are executor nodes of two classes: *stable* and *unstable*. Stable nodes are always online. Unstable nodes have two possible states (online and offline) and two transitions (from online to offline and back). Their change of state obeys some parametric, periodic or stochastic model. There are n stable nodes and m unstable nodes created in the initial simulation stage. During the simulation, unstable nodes can leave the network (causing a miss for all the tasks on their execution queue) and reconnect subsequently according to a parameterizable process. When a node comes back online it retains its identifier; in this way the behavior history of unstable nodes is preserved. Nodes are heterogeneous. Disregarding of their class they have computational resources (CPU, RAM) randomly selected in some range (uniformly distributed). The node RAM constitutes a constraint on the task that can be accepted by the node.

Simulation Duration. Our scenarios were simulated for seven hours; the end result of these simulation runs are shown in Table 2.

We have also investigated how some of the performance parameters respond over time considering a granularity of one second; this is shown in the plots 3, 4, and 5. During our experiments the transient time has proven to be less than 7 minutes.

In the following we refer to the average results obtained after reaching a 95% confidence interval, with a radius of 0.05, evaluated with the Student's t-test [7].

4.2 Performance parameters

To evaluate the performance of the proposed strategies (see Section 3) we have considered four different measurements to be relevant: the hit rate perceived, the messages spread in the network (total and refused messages), the QoS (Quality of Service) perceived by the task initiators (through the waiting and sojourn times) and the algorithm fairness (considering how well the followed approach is able to equally distribute the task load). In particular, we have considered nine different performance parameters:

1. **Hit plus running rate** (H+R): defined as the relative amount of tasks that have completed (satisfying their deadline) or are still running, over the total number of sent requests. A running task will be completed if the node that

executes it does not go offline. A higher value is desirable since it denotes 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. This performance indicator enables us to evaluate the overhead introduced by sending requests to overloaded nodes. The requests are sent until a node able to accept the request is found or the node list is exhausted. It is preferred to have low values of this parameter corresponding to a minor overhead introduced in the network for useless messages.
3. **Total number of execution requests addressed** (*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 parameter 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 the sum of the waiting and execution times. 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., it spreads the workload evenly among the nodes.
7. **The variance of tasks 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 more. This is an index of the effort by the nodes 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 in the task execution service.

Note that a $\overline{tpn}$ value close to the $\frac{\text{numberOfTasks}}{\text{numberOfNodes}}$ cannot always be obtained, e.g., if we consider the presence of a certain amount of nodes that are unstable to the extent of not being able to execute any task. Parameters $\overline{tpn}$, σ^2_{tpn}, $\max(tpn)$ and $\min(tpn)$ should be analyzed together to evaluate the strategy's ability to address the workload and to what degree it is able to evenly distribute that load.

4.3 Simulated scenarios

The archetypal scenarios we used in order to evaluate the benefit that can be achieved when reputation mechanisms are used in the node selection process to distribute the task execution requests is as follows: In our scenarios the arrival processes are Markovian, i.e., the inter-arrival time between two consecutive

Property	R	RB	PRB
H+R	0.757	0.813	0.770
Refused rate	0.519	0.020	0.004
Tot req	16742.0	25075.25	25074.666
$\overline{W}$ *time*	4520.729	4135.410	4764.227
$\overline{S}$ *time*	70404.393	71811.297	71801.275
$\overline{tpn}$	18.999	22.900	19.375
σ^2_{tpn}	119.865	175.470	127.094
max(*tpn*)	38.666	42.416	39.833
min(*tpn*)	0.25	0.0	0.0

Property	R	RB	PRB
H+R	0.550	0.482	0.551
Refused rate	0.911	0.525	0.284
Tot req	22322.916	21047.916	25435.416
$\overline{W}$ *time*	12765.934	15102.111	13214.652
$\overline{S}$ *time*	93410.104	105657.507	94366.635
$\overline{tpn}$	18.430	17.916	18.240
σ^2_{tpn}	66.430	59.491	54.883
max(*tpn*)	31.25	25.75	29.083
min(*tpn*)	1.083	1.916	1.833

Table 2. Mean task arrival: 1000 (top) and 750 (bottom)

tasks can be modeled as an exponential random variable with a mean value equal to 750 ms or 1000 ms in the comparing workload. Also, the unstable node departure and reconnection times are modeled with Markovian processes with a mean value equal to 72 seconds. The simulated time is 7 hours, and the temporal analysis considers a granularity of 200 sec. The tasks are described by a deadline of 20% of its duration, a task duration uniformly distributed in a range of [0...24] minutes and a memory requirement in the range of [0...512] MB.

4.4 Experimental results

We have considered two different task arrivals to evaluate the performance of the proposed strategies under varying workload. Table 2 shows both: First, a *low-load* scenario where tasks have a mean arrival time of 1000 ms, and second a *heavy-load* scenario where the tasks mean arrival is 750 ms.

In the low-load situation (Table 2, top), analyzing the *H+R* rate (Fig. 3) and the *Refused rate* (Fig. 4, top right), we observe that both reputation-based approaches (`RB` and `PRB`) are able to identify the stable nodes and redirect the load towards them. The *Tot req* spread in the network and queue times are almost the same for each kind of strategy. The strategies that implement a random choice (i.e., `R` and `PRB`) are able to spread the load more uniformly among the nodes (Fig. 5).

When the node load increases (Table 2, bottom), it is the `PRB` approach that obtains the best performance, since it is able to spread the load on more nodes in

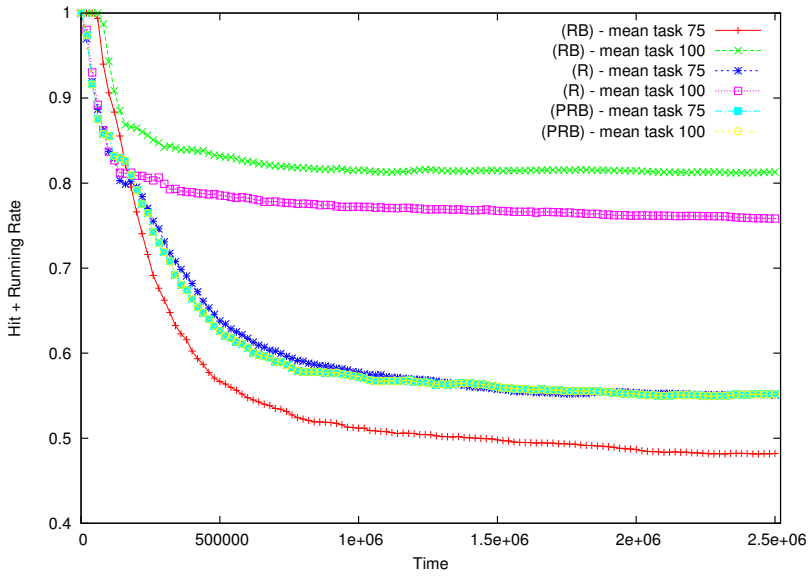

Fig. 3. *H+R* rate

comparison to the RB and the R approaches (Fig. 5). At the same time, the PRB approach is able to take into account the information gained on the evaluation of the node's behavior through the reputation scores, and does not stop on a local minimum.

Fig. 3, 4 and 5 show the benefits of using different node selection strategies. It is worth observing that the RB approach is more sensitive to the increase in system load. The RB has the greatest dependency by task load; indeed its performance on the *H+R* rate decreases quickly. The PRB approach instead is the more stable under the change of workload. The same observation arises from the *Refused rate* in the heavy load scenario, where the RB approach behaves like the random approach. The PRB approach instead shows the lowest reduction on the performance.

The $\overline{W}$ and $\overline{S}$ *times* (Fig. 4, bottom) are minimized with the RB approach where the nodes are almost able to address all the task load, since this approach is able to identify the more powerful nodes. However, when load increases and these nodes are no longer able to manage the workload, they become overloaded, which leads to a drop in queue performance and *H+R*.

The task distribution among nodes (Fig. 5) shows that the RB approach tends to direct the load to few nodes: high $\max(tpn)$ and σ^2_{tpn}.

Our conclusion is that the node selection process done through a reputation-based mechanism can be an effective way to evaluate the node behavior and thus identify the most promising nodes for task execution. Using only the reputation score, it is possible to observe a degradation on the performance when the task load is high, since the reputation initially leads to a redirection of all tasks to a few nodes that soon get overloaded and consequentially lose their score due

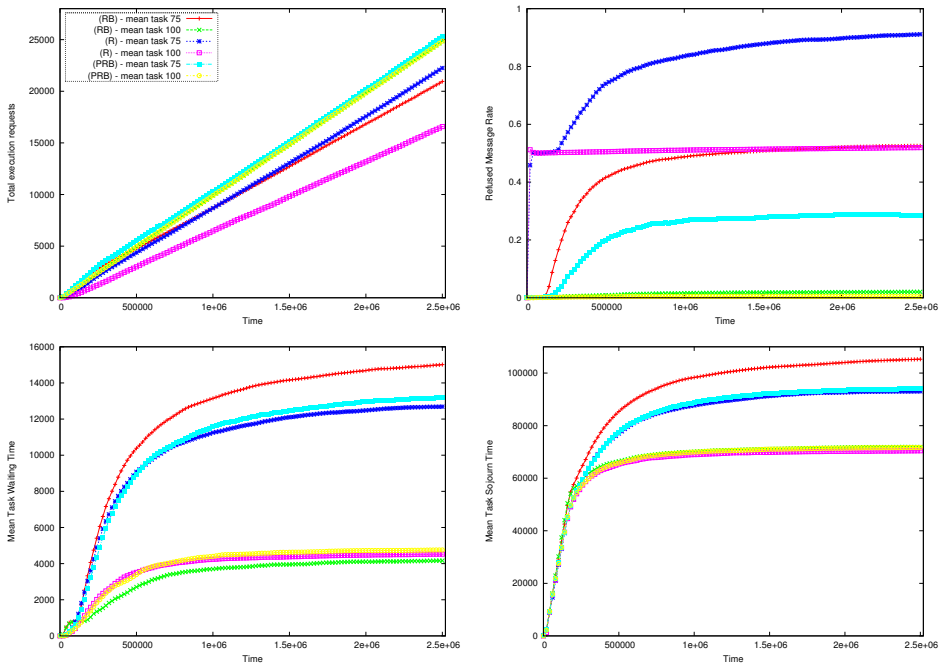

Fig. 4. *Tot req* (top left), *Refused rate* (top right), *Waiting time* (bottom left) and *Sojourn time* (bottom right).

to task rejections. The mix of the two approaches, realized in the `PRB` approach, seems to be the more effective way to use the knowledge acquired with the reputation scores, and at the same time avoids getting stuck in a performance local minimum. This is because the `PRB` approach allows some degree of exploration of the nodes that do not currently have high scores.

5 Related work

In this section, we discuss work which shares similarities with our approach or served as a source of inspiration. For the sake of brevity we focus on those works that put special emphasis on reputation, trust and load balancing aspects within the same system.

One such work is the trust management framework proposed by Mishra *et al.* [18] for the sake of trustworthy load balancing in cluster environments. The framework extends the *Jingle Mingle Model* (JMM) whose aim is to evenly distribute the task workload across multiple node within a cluster, while ensuring that the node that takes in charge the process of an overloaded node is trusted. Differently from our approach, they assume the presence of malicious nodes. Indeed, the main motivation is that, since process migration involves transferring

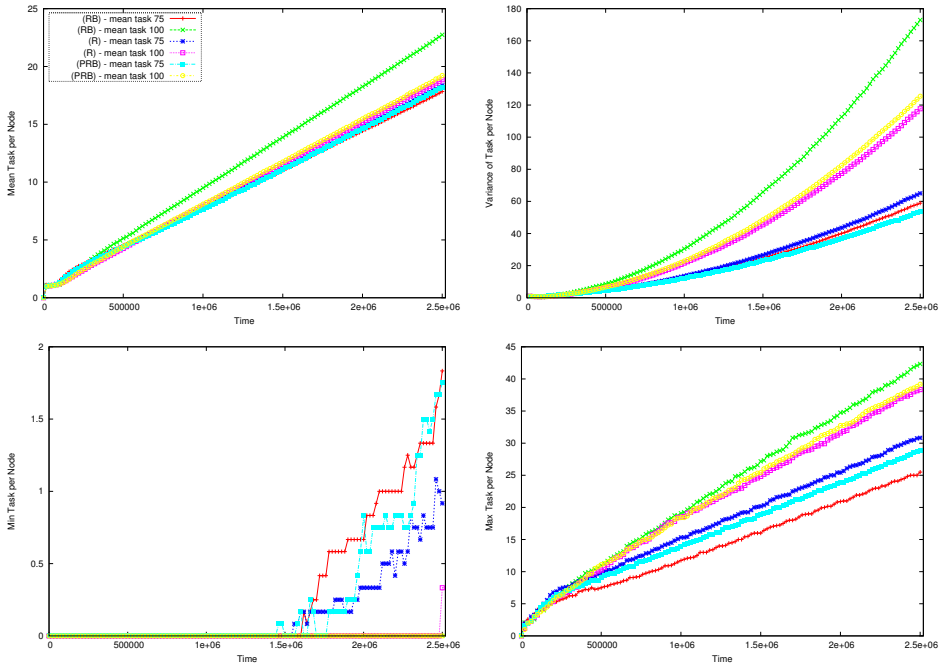

Fig. 5. *Mean* (top left), *Variance* (top right), *Min* (bottom left) and *Max* (bottom right) number of tasks.

the state of processes among nodes, a malicious node could steal or manipulate such information, which may include code, data and credentials. The proposed JMM extension considers the presence of a *Process Migration Server* (PMS), a trustworthy node in charge of authenticating nodes. It maintains a table where each row is composed by a *node id* and the corresponding trust probability. When a node joins the cluster system, the PMS assigns a trust score α equal to 0.5 to the node. Every time a node executes some remote process successfully the trust score is updated with $\alpha * n$ where n is the number of times it has successfully completed remote executions. A connection among the underloaded (*Jingle-node*) and the overloaded (*Mingle-node*) node is kept, and malicious action (e.g., the executing node inappropriately changes data) are reported to the PMS, who reduces the trust score by 1 unit. When a node's trust score goes below the threshold of 0.5 the PMS disconnects the node from the network and the corresponding table entry is removed from the table. In sum, the main difference with respect to our approach is that we use reputation scores as a measure of goodness of collaborator nodes, while their approach aims at minimizing the probability of assigning nodes to malicious nodes.

A work similar to ours is the reputation-based approach to discovery and selection of reliable resources in P2P Gnutella-like environments, proposed by Damiani *et al.* [11]. The authors propose *XRep*, a self-regulating system that

uses a distributed polling algorithm implemented in the P2P network to build a robust reputation mechanism. Each node and every resource in the P2P network has a reputation associated to a tampered resistant identifier (or to a content digest in the case of a resource). Reputations are cooperatively managed via a distributed polling algorithm to reflect the community's view on the use of a certain resource. The resource selection process in the P2P network is enriched by inquiring the network for the opinion of peers on resources and their offers. Each node maintains an *experience repository* constituted by: (i) a resource table that assigns a binary value (good or bad) to each resource it has experienced; (ii) a node table that associates to each peer the number of successful and unsuccessful downloads. These informations are shared on request. In Gnutella-like networks, when a node searches for keywords, it receives a list of resources and the corresponding nodes that offer them. Thus, XRep allows to inquire other peers about either resources and nodes that offer them. The decision from which node to start the download is based on the votes received and on the reliability of the votes (obtained through a second round of opinions on the voters from another set of peers). Our approach is inspired by this work, but we addresses a different problem: distributed QoS-constrained task execution in a cooperative environment instead of file sharing with a more stable presence of node resources. In our scenarios the focus is on both the node load, behavior (expressed as stable or unstable online presence) and node resource capability.

Many distributed computing problems similar to our case study have been approached with distributed learning-based approaches in the past. An archetypal examples are those based on *Ant Colony Optimization (ACO)*. Such approaches assume that no information about node characteristics or their load is available a priori and try to gather such information in a distributed way. The ACO approach was firstly proposed by Di Caro and Dorigo [10] and recently applied also to volunteer cloud environments, for example by Sim *et al.* [23], Mishra [17] and Ali *et al.* [1]. One crucial difference with our setting is that we assume that nodes are not willing to disclosure any information regarding the resources they own. In current work we are evaluating an ACO-based technique for volunteer clouds [21] and we are investigating how to combine it with the node *pre*-selection strategies proposed in this paper.

6 Concluding remarks

In this paper, we have investigated the problem of task distribution in voluntary, peer-to-peer cloud computing environments where nodes are willing to share their resources to other nodes. We have proposed a solution based on (1) a reputation-based approach to the collaborator selection problem, and (2) a methodology to assess, at design time, the impact of the selection strategies on the system performance. We have shown that reputation-based systems can be beneficial in cases where available node resources are unknown, or where nodes deliberately do not want to disclose their status (e.g., current load) or their resources (e.g., CPU, memory).

In our experiments, the reputation score calculated through the evaluation of node interactions has been used as the main criteria for selecting nodes for task execution. Our simulation results shows how the task performance parameters are affected by the use of three different strategies (i.e., `R`, `RB` and `PRB`).

Currently, we are calculating reputation scores by considering all aspects of the behavior of a node in a uniform way, i.e., all (satisfactory or unsatisfactory) ratings have the same weight (see Table 1). We plan to extend our analysis with more sophisticated reputation-based approaches, where separate behavioral aspects of a node (e.g., capacity or online/offline period) are rated differently and where further aspects may be taken into account. In this way, we can tune the selection strategies according the specific needs of a given cloud application, which for example may privilege node availability with respect to other features.

Furthermore, we are investigating the implementation of reputation-based node selection strategies in the Science Cloud platform to validate the simulation results with experiments on a real-world cloud platform.

References

1. A.-D. Ali and M. A. Belal. Multiple ant colonies optimization for load balancing in distributed systems. In *ICTA*, 2007.
2. M. Amoretti, M. Picone, F. Zanichelli, and G. Ferrari. Simulating mobile and distributed systems with DEUS and ns-3. In *HPCS*, 2013.
3. D. P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID*, 2004.
4. 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.
5. European integrated project ASCENS (autonomic service component ensembles). `http://www.ascens-ist.eu/`.
6. The science cloud platform. `http://svn.pst.ifi.lmu.de/trac/scp/`.
7. G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains*. Wiley, 2 edition, 2006.
8. F. Brasileiro, E. Araiijo, W. Voorsluys, M. Oliveira, and F. Figueiredo. Bridging the high performance computing gap: the ourgrid experience. In *CCGRID*, 2007.
9. J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. In S. Fitzgerald, M. Guzdial, G. Lewandowski, and S. A. Wolfman, editors, *SIGCSE*, 2009.
10. G. D. Caro and M. Dorigo. Antnet: A mobile agents approach to adaptive routing. Technical report, IRIDIA, 1997.
11. E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *CCS*, 2002.
12. Z. Despotovic and K.Aberer. A Probabilistic Approach to Predict Peers' Performance in P2P Networks. In *Cooperative Information Agents (CIA)*, 2004.
13. D. Gambetta. *Trust: Making and Breaking Cooperative Relations*, chapter 13: Can We Trust Trust?, pages 213–237. Basil Blackwell, 1988.
14. A. Jøsang and R. Ismail. The beta reputation system. In *the 15th bled electronic commerce conference*, 2002.

15. E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(1-4):72–93, 2005.
16. P. Mayer, A. Klarl, R. Hennicker, M. Puviani, F. Tiezzi, R. Pugliese, J. Keznikl, and T. Bureš. The Autonomic Cloud: A Vision of Voluntary, Peer-2-Peer Cloud Computing. In *Proceedings of 3rd Workshop on Challenges for Achieving Self-Awareness in Autonomic Systems*, pages 1–6, 2013.
17. 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.
18. S. Mishra, D. Kushwaha, and A. Misra. A cooperative trust management framework for load balancing in cluster based distributed systems. In *Recent Trends in Information, Telecommunication and Computing (ITC), 2010 International Conference on*, pages 121–125, 2010.
19. MultiVeStA website. `http://code.google.com/p/multivesta/`.
20. L. Saino, C. Cocora, and G. Pavlou. A toolchain for simplifying network simulation setup. In *SIMUTOOLS*, 2013.
21. S. Sebastio, M. Amoretti, and A. Lluch-Lafuente. A computational field framework for collaborative task execution in volunteer clouds. In *SEAMS*, 2014.
22. S. Sebastio and A. Vandin. MultiVeStA: Statistical model checking for discrete event simulators. In *VALUETOOLS*, 2013.
23. 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.
24. D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.