# A cooperative approach for distributed task execution in autonomic clouds*

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

Alberto Lluch Lafuente, Stefano Sebastio
*IMT Institute for Advanced Studies Lucca*
*{alberto.lluch,stefano.sebastio}@imtlucca.it*

***Abstract*—Virtualization and distributed computing are two key pillars that guarantee scalability of applications deployed in the Cloud. In Autonomous Cooperative Cloud-based Platforms, autonomous computing nodes cooperate to offer a PaaS Cloud for the deployment of user applications. Each node must allocate the necessary resources for customer applications to be executed with certain QoS guarantees. If the QoS of an application cannot be guaranteed a node has mainly two options: to allocate more resources (if it is possible) or to rely on the collaboration of other nodes. Making a decision is not trivial since it involves many factors (e.g. the cost of setting up virtual machines, migrating applications, discovering collaborators). In this paper we present a model of such scenarios and experimental results validating the convenience of cooperative strategies over selfish ones, where nodes do not help each other. We describe the architecture of the platform of autonomous clouds and the main features of the model, which has been implemented and evaluated in the DEUS discrete-event simulator. From the experimental evaluation, based on workload data from the Google Cloud Backend, we can conclude that (modulo our assumptions and simplifications) the performance of a volunteer cloud can be compared to that of a Google Cluster.**

***Keywords*-cloud computing; autonomic clouds; autonomous systems; volunteer computing; distributed tasks execution;**

## I. INTRODUCTION

The success of open-source cloud platforms and the wide adoption of the cloud technology in its various incarnations (SaaS, PaaS, and IaaS) by individual users, public institutions and private companies, is opening new scenarios where autonomous cloud entities (e.g. cloud sites) can interact with each other. Prominent examples are the Volunteer Computing paradigm [1], and its various evolutions [2], [3], which promote the idea of what may be called *Autonomous Cooperative Cloud-based Platforms* (ACCPs).

The provisioning of cloud-based application execution services in ACCPs poses several challenges: (i) The execution service is subject to Service Level Agreements (SLAs) that impose QoS requirements to be monitored and optimized; (ii) Optimization criteria are critical for both the provider (who may have policies like minimizing the number of active resources) and for the customer (who may be interested in the best QoS/price ratio); (iii) ACCPs are *inter-cloud* systems made of several, heterogeneous autonomous clouds, each with its own resources, goals, management policies and rules which may even forbid some forms of cooperation (see e.g. [4]); (iv) ACCPs are highly dynamic and open: participants can leave and join, the performance of applications may vary. Self-adaptive mechanisms are crucial to tackle such dynamism; and (v) Resource-awareness is crucial to devise flexible and performant solutions: PaaS entities need to cooperate with their underlying IaaS.

Each participant of an ACCP must allocate the necessary resources for customer applications to be executed with certain QoS guarantees. If the QoS of an application is compromised a node has mainly two options: to use more computational resources (e.g. requesting the underlying IaaS layer to startup a new VM) or to rely on the collaboration of other nodes (e.g. requesting the remote execution of part of the application). This decision is not trivial since it involves many factors, including the cost of setting up virtual machines, migrating applications and discovering collaborators.

**Contribution.** We propose a model for ACCPs instantiated and validated in a concrete case study (namely, the SCIENCE CLOUD [3]), together with the evaluation of a cooperative approach. The experimental evaluation is supported by DEUS [5] a general-purpose, discrete event simulator, that has been successfully applied for the modeling and simulation of various other complex systems. Our evaluation exploits real workload data from the Google Cluster dataset [6] and provides estimations of the performance of various cooperation strategies for different SCIENCE CLOUD configurations showing that collaborative strategies tend to perform better than selfish ones, in particular for large number of nodes.

**Synopsis.** §II introduces our case study. §III presents our model of ACCPs, describing some of the main ingredients we envisage to face the challenges (i–v) mentioned above, with a focus on our case study. §IV reports our experimental evaluation. §V summarizes some recent research efforts regarding task mobility, interoperability and agent-based architectures for cloud computing. §VI includes some concluding remarks and outlines our current and future research efforts.

## II. CASE STUDY: THE SCIENCE CLOUD

The SCIENCE CLOUD [3] is an ACCP being developed within the European project ASCENS [7] with the aim of

*Research supported by the European Integrated Project 257414 ASCENS.

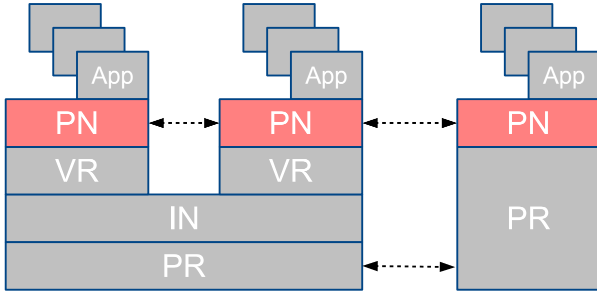


Figure 1. The SCIENCE CLOUD architecture.

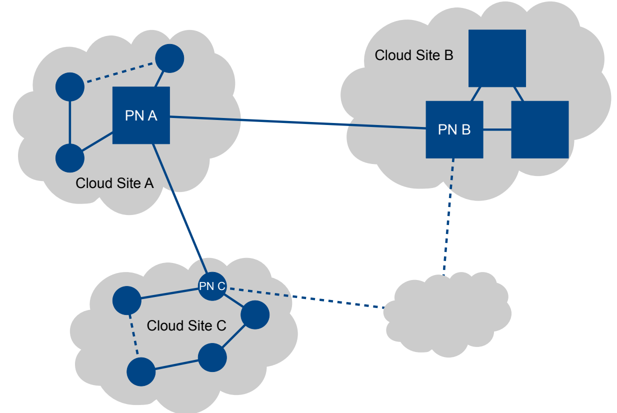


Figure 2. Example of ACCP network.

creating a decentralized platform for sharing computational resources in scientific communities. It provides distributed application execution as its main functionality. Participants contribute with their desktops, mobile devices, servers, or virtual machines by running *Platform Nodes* (PNs) on them. PNs communicate with each other and may form aggregations (called *ensembles*) to collaborate in order to run their applications in a robust and performant way, so to satisfy their QoS constraints.

Although the SCIENCE CLOUD is essentially a sort of volunteer computing platform, there may be autonomous parts under the control of commercial or academic entities. This means that each PN may have different goals or policies in what regards the use and sharing of computational resources.

Fig. 1 schematizes an instance of the SCIENCE CLOUD with three PNs . One of the PNs runs over ordinary physical resources (PR), while the rest rely upon a possibly shared cloud infrastructure providing virtual resources (VR). Indeed, the SCIENCE CLOUD can be seen as a PaaS, whose nodes possibly run on top of IaaS nodes (INs).

A PN is a volunteer node that decides the amount of resources it wants to share in the network and based on this decision, creates an application environment like a sandbox or a virtual machine (VM). PNs running on top of an IN can resize their application environments, for instance by scaling-up the VMs (actually, the case we have considered here) or to instantiate other VMs. The rest of the PNs cannot resize their application environments (as shown in Fig. 1).

The main role of a PN is to provide a service for executing applications. In this paper we assume applications to be sets of independent tasks to be executed with a certain QoS, specified by a SLA. In particular, we consider SLAs that impose a certain deadline for the execution of each task.

The adaptivity logic of each PN should take care of reacting to conditions that may compromise the SLA of the applications, such as overload or shutdowns. The adaptivity logic can act on two levels: (i) on the platform level, by migrating part of the applications to other PNs (thus balancing load or compensating for lost PNs), and (ii) on the infrastructure level, by requesting new resources (e.g. VMs) to adapt to an excess of load. The latter is of course only available for PNs running over an IN (most typically a server, rather than mobile devices) with a suitable actuator interface.

In order to take the right decisions, the adaptivity logic relies on the knowledge each PN has, which consists of what the PN knows about itself, about its infrastructure (e.g. CPU load, available memory), and about other PNs (e.g. acquired through the network).

In order to cooperate, PNs communicate over the network. The protocol followed by these communications must enable PNs to find one another and establish links, for example by means of discovery mechanisms. Furthermore, PNs must be able to query others for knowledge and distribute their own. Finally, the protocol must support exchange of data and applications.

## III. MODEL

We present here a modular and extensible model of ACCPs, exemplified with the SCIENCE CLOUD case study. Our presentation follows the structure of the system illustrated in Fig. 1: applications are executed on top of PNs, which rely either on virtual resources provided by an underlying INs or directly on physical resources. PNs communicate over a platform-level overlay network that relies upon an underlying network such as the Internet.

Here we focus on some of the challenges outlined in Section I, we will consider also the rest in a future work.

Each PN running on top of an IN is modeled as a $G/G/m/K$ queue, where $G$ means that task arrivals and service times have generic distributions, $m$ is the number of VMs/PMs in each node, and $K$ is the maximum number of tasks in the system (1 being executed, $K-1$ waiting in an FCFS queue) [8]. The rest of the PNs are modeled as $G/G/1/K$ queues since they cannot resize their resources.

### A. Applications

We model applications as sets of independent *tasks*, defined by their duration and their degree of parallelism. Our model

assumes that tasks durations are fixed and that nodes are able to perfectly predict their duration. Moreover, we assume a perfect linear speed-up for the parallelism.

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

A degree of parallelism $\rho$ means that the a task with duration $\delta$ can be ideally executed in a CPU with $\rho$ computational units (e.g. processors) in time $\delta/\rho$.

Actually, we assume that applications generate *task execution requests* to the underlying PaaS platform. In our model task execution requests are defined by a task and a termination deadline (from the SLA).

*Definition 3.2 (task execution request):* A *task execution request* is a tuple $\langle \delta, \rho, \mu, \tau_a, \tau_d \rangle$, where $\langle \delta, \rho, \mu \rangle$ is a task, $\tau_a \in \mathbb{R}+$ is the task arrival date, and $\tau_d \in \mathbb{R}+$ is its termination deadline.

## B. Virtual and Physical Resources

We model both virtual and physical resources uniformly as VMs. The main feature of such VMs will be their CPUs as our main concern is the running time of applications, which we consider to be compute intensive (rather than data- or communication-intensive), i.e. their running time depends essentially on the amount and properties of the CPUs.

*Definition 3.3 (CPU):* A *CPU* is defined by its frequency $\phi \in \mathbb{N}^+$, expressed in GHz.

We consider resources to be machines modeled just as sets of homogeneous CPUs (that is, with identical frequency and number of cores).

The execution time $e(T, M)$ required for completing a task $T = \langle \delta, \rho \rangle$ on a machine $M$ whose CPUs have frequency $\phi$ is defined by the following equation:

$$e(T, M) = \frac{\delta}{\phi \cdot min\{\rho, |M|\}} \quad (1)$$

That is, we divide the duration of one single CPU ($\delta/\phi$) by the maximum degree of parallelism that can be exploited, which is bounded by both the amount of available CPUs ($|M|$) and the parallelism degree of the task ($\rho$).

A task can be executed on a machine only if the memory requirement constraint is satisfied:

$$\mu \leq RAM(M) \quad (2)$$

where $RAM(M)$ represents the amount of available memory on a machine $M$.

A machine $M$ accepts a task $T$ in its queue only if it can respect the task deadline.

These equations are used not only as a model of actual execution time, but also as the model that PNs use in their estimations (see Section III-E).

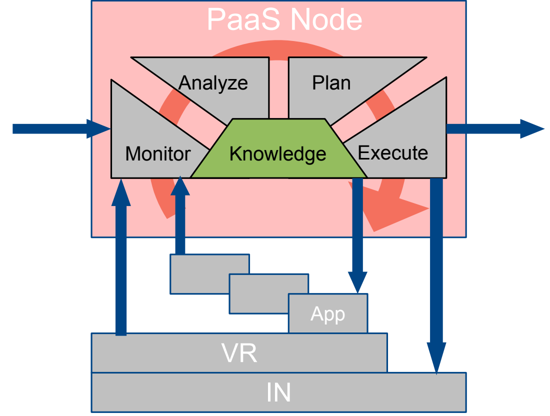


Figure 3. Architecture of a PN.

## C. Network

In general, PNs can communicate with each other in arbitrary ways as they are all connected to the Internet. Of course, the need to identify PNs willing and able to share part of the application load calls for adequate logical structures such as overlay networks. At the moment we are considering a simple overlay network that exploits the structure of ACCPs.

In particular, we assume that ACCPs are structured into *cloud sites* as illustrated in Fig. 2. Each PN resides on exactly one site. Some PNs (denoted with squares) can be considered as stable nodes (e.g. data centers) while others (denoted with circles) are nodes that can dynamically join or leave the platform (most typically, PNs without an underlying IN).

PNs are organized in hierarchical overlay networks, where nodes of different cloud sites can establish a direct communication links, with the mediation of *supernodes*.

If a cloud site contains an PN running on top of an IN such as a data center, it will act as a supernode for the rest of the nodes of the same cloud site. The rationale is that to such PN can enjoy better uptime and resources.

Cloud sites without data centers are organized in random topologies. Several strategies can be used to select super nodes of such sites: from classical leader election strategies to more specific ones. For example, an idea could be to use strategies that give more chances to be elected to sites with longer online presence (i.e. more stable nodes) or those who share more resources (i.e. the more collaborative nodes).

We assume that the time required by ordinary communication (e.g. sending requests and replying) can be neglected. Instead, we assume that the time needed to migrate a task from a PN in Cloud Site $u$ to a PN in Cloud Site $v$ depends on the distance between $u$ and $v$.

## D. IaaS Nodes

INs are essentially modeled by a set of physical resources. They have no particular logic, except for to faithfully serve the

requests of the PNs they support: i.e. providing them virtual machines built out from their pool of physical resources.

### E. PaaS Nodes

As already mentioned, PNs can be supported or not by an underlying IN. PNs without infrastructure correspond to participants using mobile devices or personal desktops that join and leave the platform in a volunteer manner and their presence is not assured during time. PNs with infrastructure are stable nodes consisting of cloud-based data centers that participate to the platform in a permanent manner.

Internally, PNs are structured according to IBM's reference architecture for autonomic systems [9], as illustrated in Fig. 3. Brifely, each PN monitors applications, resources and the environment, analyzes its status, devises plans to improve its execution, and enacts those plans. A Knowledge base supports this cycle of activities.

*Monitor and knowledge:* The incoming arrows in Fig. 3 denote the sources of information that the monitoring activity uses to enrich the knowledge. In particular, PNs monitor (i) the load of their resources (e.g. the load of virtual or physical machines); (ii) the performance perceived by the application (e.g. ratio of task execution requests whose SLA has been satisfied), and (iii) its environment (e.g. its partner PNs). All the monitored data is collected into the knowledge base.

*Effectors:* PNs can act at three layers as denoted by the outcoming arrows in Fig. 3. At the infrastructure layer, new resources can be requested to the underlying IN (if present). At the platform layer, remote PNs can be requested for executing tasks. At the application layer, tasks can be migrated to other PNs.

*Analyze/Plan:* The analysis and planning activities have to deal with two major issues: the allocation of local resources to execute tasks locally and the distribution of tasks among collaborating PNs.

Regarding the first issue, we assume that PNs provide exclusive, isolated application execution environments by scheduling the execution of tasks or sub-tasks sequentially. Of course we can use more complex scheduling mechanisms but a comparison of different scheduling strategies is out of the scope of this paper.

We consider different degrees of cooperation among PNs. A *selfish* PN sends requests for remote task execution when local resources are not sufficient, but rejects all external requests. Instead, a *volunteer* PN always accepts external requests, when locally available resources are sufficient. We assume that each PN has a set of *known* PNs in its knowledge base. Several strategies for building and maintaining such knowledge can be found in the literature but we assume here a simple one based on a fixed pre-defined set.

In between the two opposite strategies, there is a partial volunteering scheme for which a PN may decide to accept or reject a task execution request, depending on the result of the following the test:

$$currentMissRate \leq MissRateTolerance \quad (3)$$

where the $MissRateTolerance$ constitutes the *volunteering degree* and the $currentMissRate$ is the current rate of discarded tasks due to the inability to fulfill their deadlines. The *MissRateTolerance* is used by PNs to ensure their own QoS. Clearly, the extreme miss rates of $0$ and $+\infty$ correspond to the selfish and volunteer strategies, respectively.

## IV. Simulations

This section presents our model implementation and evaluation by means of the discrete event simulator DEUS, which is shortly described in §IV-A. Next (§IV-B) we describe the configuration of the scenario under evaluation. The rest of the section is devoted to reporting and discussion of results. All our experiments have been run on an a laptop equipped with a 2.0 Ghz Quad Core CPU and 8 GB of RAM. Our simulator and the configuration files of the experiments are available for download at http://code.google.com/p/deus/downloads/list.

### A. DEUS in a nutshell

Several free or commercial discrete event simulation tools are available, but no one appears to be sufficiently generic and flexible to support the analysis of complex systems at every scale such as the ones under study in this paper. DEUS [5] is a novel general-purpose, open-source simulation environment, characterized by extreme ease of use and flexibility.

DEUS is multi-platform, being developed in Java. Its API allows developers to implement (by sub-classing) (i) *nodes*, i.e. the entities which interact in a complex system, leading to emergent behaviors such as humans, pets, cells, robots or intelligent agents; (ii) *events*, e.g. node births and deaths, interactions among nodes, interactions with the environment, logs and so on; and (iii) *processes*, either stochastic or deterministic ones, constraining the timeliness of events.

A node may represent a dynamic system characterized by a set of possible states, whose transition functions may be implemented either in the source code of the events associated to the node, or in the source code of the node itself. Multi-scale modeling of complex system can be achieved by defining nodes of different complexity and connecting them. DEUS comes with a library of predefined, common processes, and many others can be implemented by the user.

### B. Simulated scenario

We describe here the main characteristics of the scenario used in the experiments. We focus on two main aspects: the network and the workload models. As a matter of fact each simulation run has two stages: one for generating the network configuration (cloud sites and participants), and another one for evaluating the actual activity period.

The configuration of the Science Cloud includes a variable number (from $1,000$ to $10,000$) of stable PNs, each

Table I
DISTRIBUTION OF TASK ATTRIBUTES

| size | duration (hours) | CPU (cores) | RAM (GBs) | Deadline max offset (minutes) | Poisson mean arrival (milliseconds) |
|---|---|---|---|---|---|
| small | $0 - 0.4$ | 1 | $0 - 0.5$ | 5 | 200 (20% of max duration) |
| large | $1 - 12$ | $1 - 4$ | $1 - 4$ | 288 | 600 (40% of max duration) |

Table II
DISTRIBUTION OF NODE ATTRIBUTES

| type of node | CPU frequency (GHz) | CPU (cores) | RAM (GBs) | Number of nodes |
|---|---|---|---|---|
| PN without IN | $1 - 2$ | $1 - 6$ | $0.1 - 2$ | $1,000 - 10,000$ |
| PN with IN | $1 - 3$ | $2 - 32$ | $2 - 6$ | 7 (in different sites) |

one having an identifier that specifies its cloud site. Every site is managed by a PN (possibly running on top of an IN) which is elected as supernode. The supernode receives task execution requests from PNs of its cloud Site, and from the supernodes of remote cloud Sites. Each supernode maintains an index of the PNs of its cloud site, which are in charge of notifying the supernode of their status (online, going offline).

We consider the presence of 10 different cloud Sites, 7 of which are managed by an IN supernode and the others rely only on PNs. The PNs that are not on top of INs are less computationally powerful (see Table II). The cost for transferring a task between two neighbor Cloud Sites (i.e. the cost of a hop) is set to 1% of the maximum duration of a small task. In this manner also small tasks have the possibility to be transferred to another Site. We consider the communication overhead for data transferring within a Cloud Site to be negligible.

For the workload characterization, our main reference is the Google Cloud Backend [6] (arguably the largest cloud backend on the planet) described in [10]. There, tasks are characterized according to their duration, CPU requirements and memory requirements, each abstracted as either *small* (s) or *large* (l). Typical examples of long-running tasks are user-facing ones (which run continuously so as to respond quickly to user requests) and compute-intensive ones (such as processing web logs). Examples of short-running tasks are highly parallel operations such as index lookups, searches and Map-Reduce operations. We have considered that in the Map-Reduce paradigm the short tasks (Map-task) are mainly the result of a single parallelized job, the Reduce operation cannot be done if every Map-task has not terminated its execution. Thus, tasks that belong to a parallelized operation must finish sooner, not to lose the advantage of parallelism, therefore we have assumed that short tasks have a more stringent deadline requirement. In the Google Cloud Backend, the task attributes are uniformly distributed within the intervals (grouped by qualitative coordinates) reported in Table I. Tasks with short duration dominate the task population (being the 66%).

The Google Cluster dataset [6] provides obfuscated information about the real hardware characteristics of Google cluster nodes: every reported value is normalized to the capacity of the best cluster node. Another obfuscated information relevant for the purpose of our work regards QoS properties such as deadlines. For these reasons we have made some assumptions. We consider that the CPUs of the cluster used have a frequency of 1 GHz. Table I summarizes the task characteristics.

The duration of the simulated scenario is of 7 hours (with a granularity of milliseconds), the same of the Google Cluster traces dataset [6].

Arrival processes are Markovian, i.e. the interarrival time between two consecutive tasks can be modeled as an exponential random variable with mean value equal to 100 ms for large tasks and 50 ms for small tasks. The simulated scenario considers the queue models we mentioned in § III, namely PNs running on top of an IN are modeled as $M/G/m/+\infty$ queues, while the rest of the PNs are modeled as $M/G/1/+\infty$ queues. A task is accepted for execution by a PN only if the latter is able to guarantee its completion within its deadline, otherwise the task is discarded. A completed task marks a hit for the PN on which it has been executed.

Following the above arguments, we have assumed that small tasks are more latency dependent, thus they have a deadline equal to the 20% of their maximum duration. Instead, long tasks are less restrictive on their execution, so their deadline is the 40% of their duration.

We considered the following performance parameters (an adapted selection from [11]).

- Hit + Running Rate —Defined as the relative amount of tasks that have completed or are still running with the certainty that their deadline will be satisfied. The Hit + Running Rate measures how good the system is satisfying the QoS requirements imposed by the users. We measured the Hit + Running Rate for small tasks, large tasks and both types together.
- Rate of Refused Requests for Remote Execution — Defined as the relative amount of refused requests over the total number of sent requests. This performance indicator measures the overhead introduced by sending requests to overloaded or selfish PNs, until a PN willing to accept the request is found.

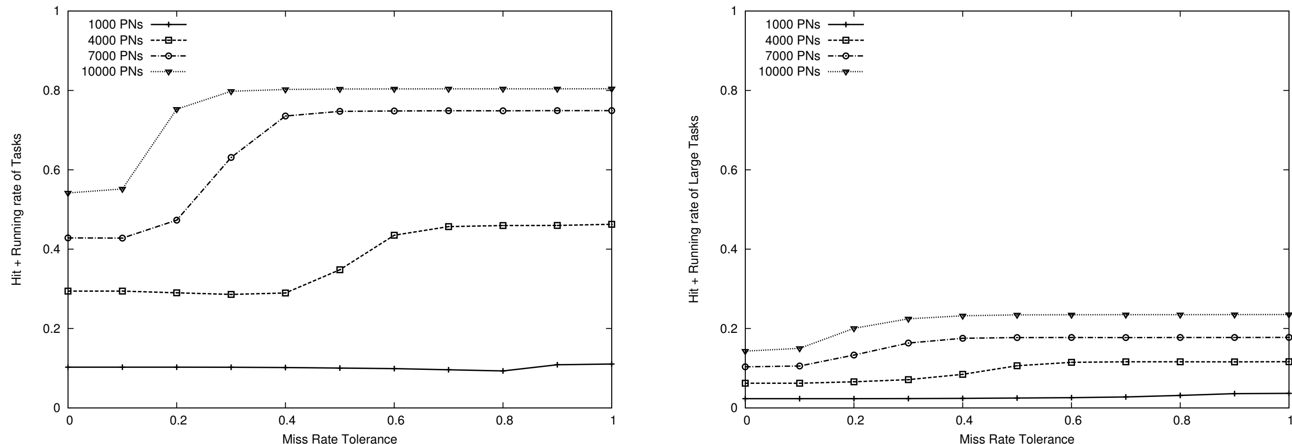


Figure 4. Hit + Running Rate of tasks (left) and large tasks (right), for different values of the miss rate tolerance and with different numbers of PNs.

### C. Results

In the following we have discussed the mean results obtained after five simulations where the variances affect only the third decimal digit. We first measured the effect of changing the *miss rate tolerance* in the cooperation strategy.

Apart from the basic common configuration we described above, it is worth mentioning that every PN uses the same strategy with exactly the same miss rate tolerance (although the DEUS simulation tool allows to define groups of peers with different strategies). We have performed parametric simulations, to study the behavior of the system for different strategy configurations.

In Fig. 4 (left), 4 (right) and 5 (left) we report the Hit + Running Rate for all, large, and small tasks, respectively. The values are plotted considering the miss rate tolerance on the horizontal axis. Each graphic includes plots for different numbers of PNs.

Obviously, the higher the number of nodes, the better the performance of the system is in terms of Hit + Running Rate. The results show the superiority of the volunteer approach. We can also observe that almost all the small tasks are completed, when the number of PNs is equal or higher than $7,000$ and the cooperation strategy tends to the volunteer approach. Large tasks have more difficulties in finding an adequate PN that can execute it, due to their higher requirements in terms of CPU cycles. The overall performance is acceptable because small tasks are the majority.

In all cases we can see that the performance in terms of hit rate is almost stabilized, starting from a miss rate tolerance equal to $0.3$. For higher values, the performance is not appreciably improved.

In Fig. 5 (right) we report the Rate of Refused Requests for Remote Execution. We see that generally the number of refused requests is lower, when the number of PNs increases. This happens because, by increasing the number of PNs, it is more likely to find an unloaded node that can accept the remote request for execution. In this figure there is not a single point of split-up related to a value of miss rate tolerance (like in the previous three figures). However, over $7,000$ PNs the number of refused requests increases because the added PNs are not those that are not on top of INs, *i.e.* they might be able to execute small tasks but in general they might not be able execute large ones. The number of INs is 7 in all experiments, as described in the beginning of Sec. IV-B. Thus, increasing the number of PNs without IN means decreasing the likelihood to find a node able to execute large tasks and hence increasing the number of request refusals for such tasks.

In a second experiment, we measured the impact of changing the *migration cost*, *i.e.* the cost for remote execution among sites. We have considered different values for the migration cost with the only constraint of still allowing to execute tasks in the most far site. Recall that the deadline of small tasks is 10% of their maximum duration. The parameter of migration cost is varied throughout the maximum duration of small tasks with an increasing step of 0.05% up to 0.25%. The results show that the hit rate is only slightly affected by the variation of migration cost. The performance on the refused message rate follows the behavior of the previous three plots.

We also measured the impact of varying the *deadline*. Obviously, by relaxing the deadline we augment the likelihood for a PN to execute the task while respecting its QoS requirements. As we already said, in the previous two experiments we considered a deadline of 0.2% (288 seconds) of the maximum duration for small tasks and of 0.4% (17,280 seconds) for large tasks. We have changed these parameters with linear increments of the amount of the base case, for the two types of tasks. For small tasks this means that we set the deadlines to 0.2%, 0.4%, 0.6% and so on. For large tasks instead we set the deadline factors to 0.4%, 0.8%, 1.2%,

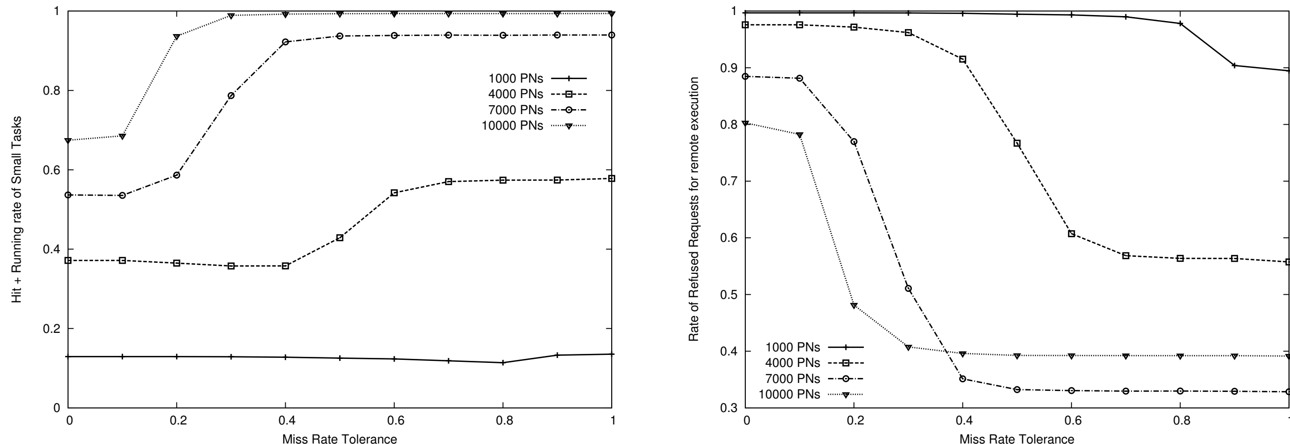


Figure 5. Hit + Running Rate of Small tasks, for different values of the miss rate tolerance and with different numbers of PNs (left), and Rate of Refused Requests for Remote Execution of the tasks, for different values of the miss rate tolerance and with different numbers of PNs (right).

etc. We obtained only a slight, less significant increment of the hit rate. The reason is that, as we showed in the first experiment, the system has just reached the saturation of satisfaction on small tasks and misses are mostly due to lack of memory on PNs to execute large tasks. They hence do not benefit for milder deadlines. The behavior for the rate of refused requests follows the ones showed in the previous two experiments.

## V. Related work

One of today's main concerns for cloud-based service providers is the increasing number of world-wide application service consumers. To deal with this problem, IaaS providers have established data centers in multiple geographical locations to provide redundancy and ensure reliability in case of site failures. For example, Amazon has data centers in the US (*e.g.* one in the East Coast and another in the West Coast) and Europe. Recently, the idea of introducing seamless, automatic mechanisms for scaling hosted services across those geographically distributed data centers has become popular, not only in theory but also in practice. For example, Amazon EC2 Spot Instances allow customers to bid on unused Amazon EC2 capacity and run those instances for as long as their bid exceeds the current Spot Price, that changes periodically based on supply and demand. However, such *Inter-Cloud* [12] raises many challenges concerning federation, security, interoperability, vendors' lock-ins, trust, legal issues, QoS, monitoring, and billing.

Such challenges are already under investigation by the research community. Our approach to cooperation strategies and autonomic mechanisms in ACCPs is our humble contribution to this field, but there are of course many other approaches that are worth considering. Due to lack of space we mention here the most relevant to our approach, focusing on three of its differentiating features: (i) task mobility, (ii) interoperability, and (iii) agent-based architecture.

Some researchers are already approaching the technical issues in managing the entire bundle of services, providing Routing-as-a-Service [13] and of moving VMs over wide area networks without losing service [14].

Erdil [11] proposes information proxies as an enabling mechanism for interoperability in federated clouds, because of two equally important characteristics of proxies: (i) ability to bypass administrative barriers, and (ii) capability to improve dissemination performance with respect to various performance parameters. Moreover, the author shows that using no hierarchical organization and picking proxy nodes randomly leads to effective system behavior, measured based on several success criteria, including dissemination overhead, redundant message ratio, and average edge count.

The Clouds-Using-Agents paradigm [15] has been advocated as a convenient way to tackle the complexity of cloud systems. Examples of this paradigm applied to service management in Inter-Cloud systems include Reservoir [16], Cloudbus [12], and SORMA [17]. In Reservoir, individual Service Manager agents interact co-operatively towards a Service Manifest. In Cloudbus, cloud brokers interact with the cloud coordinator. In SORMA, cloud brokers interact with bidders and sellers.

One step beyond, Mearns *et al.* [18] propose a bundled service provider agent architecture, which can negotiate on the open service market. This approach aims to also optimize the utilization of the providers infrastructure, while reducing the risk of failure to users through total service management.

## VI. Conclusions and Future Work

In this paper we highlighted the advantages of cooperative approaches in Inter-Cloud environments, such as ACCPs. We proposed a general architecture and its simulation-based evaluation, based on the Science Cloud [3] case study under a model of realistic workload taken from the Google Cluster dataset [6].

Our simulator can be used to estimate the performance of ACCPs either at run-time (for prediction purposes) or at early-prototyping phases (for designing purposes). For instance, from the presented experimental evaluation we can conclude that (modulo our assumptions and simplifications) a volunteer cloud with $7,000$ PNs may obtain a comparable performance to the Google Cluster consisting of $11,000$ nodes with higher capacity.

We believe that one of the main factors that impact the performance of volunteer approaches is the dissemination of load information, which helps overloaded PNs to quickly identify the most promising PN when looking for support. Therefore, we are investigating solutions based on semistructured peer-to-peer overlay networks, which provide efficient node discovery and knowledge sharing. We will also investigate more sophisticated and efficient decentralized task allocation strategies.

In the current version of our work we have considered that all tasks are independent. Tasks dependencies can lead to different and non trivial problems such as more stringent deadline constraints (e.g. some tasks may not be scheduled if other tasks have not finished their execution). We plan to add to our simulator the concept of *job* or *workflow*, that relates one or more tasks, possibly following the *Google Cluster Workload Traces* but also considering more general applications. In a future work we shall consider more detailed and sophisticated models, in particular for which regards parallelism issues and task durations. Regarding the latter we will use different task duration models (e.g. according to stochastic distributions) and prediction capabilities.

The analytical modeling of P2P network dynamics is indeed a complex task, and in what regards P2P-based cloud systems, we are taking into account the recent and relevant literature.

We have planned, as a future work, to refine our model according to real testing with the SCIENCE CLOUD [3]. Moreover, we will refine the model and its realization in the DEUS simulator, by adding different P2P dynamics, Cloud-related costs (e.g. the time to put on and tear down a VM), and with a comparison of different task distribution strategies. Additionally, we will consider the impact on performances of different supernode election strategies.

## REFERENCES


[1] D. P. Anderson, "Volunteer computing: the ultimate cloud," *ACM Crossroads*, vol. 16, no. 3, pp. 7–10, 2010.

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

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

[4] "What is google app engine?" https://developers.google.com/appengine/docs/whatisgoogleappengine.

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

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

[7] "European integrated project ASCENS (autonomic service component ensembles)," http://www.ascens-ist.eu/.

[8] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi, *Queueing Networks and Markov Chains*, 2nd ed. Wiley, 2006.

[9] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, 2003.

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

[11] D. C. Erdil, "Autonomic cloud resource sharing for intercloud federations," *Future Generation Computing Systems*, 2012.

[12] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010)*, 2010, pp. 13–31.

[13] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[14] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 67–74, 2010.

[15] D. Talia, "Cloud computing and software agents: Towards cloud intelligent services," in *WOA*, ser. CEUR Workshop Proceedings, G. Fortino, A. Garro, L. Palopoli, W. Russo, and G. Spezzano, Eds., vol. 741. CEUR-WS.org, 2011, pp. 2–6.

[16] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 1–11, 2009.

[17] J. Nimis, A. Anandasivam, N. Borissov, G. Smith, D. Neumann, N. Wirstrom, E. Rosenberg, and M. Villa, "Sorma - business cases for and open grid market: Concept and implementation," in *5th international workshop on Grid Economics and Business Models (GECON '08.)*, 2008, pp. 173–184.

[18] H. Mearns, J. Leaney, A. Parakhine, J. Debenham, and D. Verchere, "An autonomic open marketplace for inter-cloud service management," in *4th IEEE International Conference on Utility and Cloud Computing (UCC 2011)*, 2011, pp. 186–193.