Cluster Comput (2014) 17:79–100
DOI 10.1007/s10586-013-0283-6

SpeQuloS: a QoS service for hybrid and elastic computing infrastructures

Simon Delamare · Gilles Fedak · Derrick Kondo · Oleg Lodygensky

Received: 1 October 2012 / Accepted: 29 May 2013 / Published online: 28 June 2013

Abstract The large choice of Distributed Computing Infrastructures (DCIs) available allows users to select and combine their preferred architectures amongst Clusters, Grids, Clouds, Desktop Grids and more. In these hybrid DCIs, elasticity is emerging as a key property. In elastic infrastructures, resources available to execute application continuously vary, either because of application requirements or because of constraints on the infrastructure, such as node volatility.

In the former case, there is no guarantee that the computing resources will remain available during the entire execution of an application. In this paper, we show that Bag-of-Tasks (BoT) execution on these "Best-Effort" infrastructures suffer from a drop of the task completion rate at the end of the execution.

The SpeQuloS service presented in this paper improves the Quality of Service (QoS) of BoT applications executed on hybrid and elastic infrastructures. SpeQuloS monitors the execution of the BoT, and dynamically supplies fast and reliable Cloud resources when the critical part of the BoT is executed. SpeQuloS offers several features to hybrid DCIs users, such as estimating completion time and execution speedup. Performance evaluation shows that BoT executions can be accelerated by a factor 2, while offloading less than 2.5 % of the workload to the Cloud.

We report on several scenarios where SpeQuloS is deployed on hybrid infrastructures featuring a large variety of infrastructures combinations. In the context of the European Desktop Grid Initiative (EDGI), SpeQuloS is operated to improve QoS of Desktop Grids using resources from private Clouds. We present a use case where SpeQuloS uses both EC2 regular and spot instances to decrease the cost of computation while preserving a similar QoS level. Finally, in the last scenario SpeQuloS allows to optimize Grid5000 resources utilization.

Keywords Distributed computing infrastructures · QoS · Grids · Cloud

1 Introduction

Bag of Tasks (BoT) applications represent a significant part of scientific computing workload. In the same time, infrastructures to execute these BoTs tend to diversity. Depending on parameters such as performance, reliability, cost or quality of service, scientific communities can choose their infrastructure amongst Clusters, Grids, Clouds, Desktop Grids and more, and also use any combinations of them.

An emerging characteristic of these infrastructures is elasticity. In elastic infrastructures, the set of available resources can change during time. For instance, Cloud computing allows to scale up or scale down resources according to users' needs. The other source of elasticity is caused by constraints from the infrastructures. For instance, Best Effort DCIs (BE-DCIs) are an infrastructure or a particular us-

S. Delamare
CNRS, University of Lyon, Lyon, France
e-mail: Simon.Delamare@ens-lyon.fr

G. Fedak (✉)
INRIA, University of Lyon, Lyon, France
e-mail: Gilles.Fedak@inria.fr

D. Kondo
INRIA, University of Grenoble, Grenoble, France
e-mail: Derrick.Kondo@inria.fr

O. Lodygensky
IN2P3, University of Paris XI, Paris, France
e-mail: Oleg.Lodygensky@lal.in2p3.fr

age of an existing infrastructure that provides unused computing resources without any guarantees that the computing resources remain available to the user during the complete application execution.

Desktop Grids (Condor [26], OurGrid [6], XtremWeb [16]) and Volunteer Computing Systems (BOINC [4]), which rely on idle desktop PCs are typical examples of Best Effort DCIs. An example of elasticity in Grid computing is usage of a best effort queue to harvest idle nodes in a cluster, implemented by resource managers such as OAR [11]. Tasks submitted in the best effort queue have the lowest priority; at any moment, a regular task can steal the node and abort the on-going best effort task. Cloud computing users can also be subject to infrastructure elasticity. When Amazon EC2 Spot instances [2] are used, the resources availability depends on the evolution of market price. Other Cloud services [30] implement similar concept. This is a relevant example of Cloud usage as a Best Effort DCI.

Although these BE-DCIs are prone to node failures and host churn, they are still very attractive because of the vast computing power provided at an unmatched low cost. Unsurprisingly, several projects such as EDGeS [37] or SuperLink [17] have built hybrid DCIs, where Desktop Grids, can be used in conjunction with Grids and Clouds.

The drawback of BE-DCIs is their low reliability, and they offer poor Quality of Service (QoS) with respect to traditional DCIs. This study presents how the execution of BoTs, which are the most common source of parallelism in Grid Computing [31], is affected by the unreliable nature of BE-DCIs: the main source of QoS degradation in BE-DCIs is due to the *tail effect* in BoT execution. That is, the last fraction of the BoT causes a drop in the task completion throughput.

To enhance QoS of BoT execution in BE-DCIs, we propose a complete service called SpeQuloS, which abbreviates "Speculative execution and Quality of Service". SpeQuloS improves the QoS in three ways: (i) by reducing time to complete BoT execution, (ii) by improving BoT execution stability and (iii) by informing user about a statistical prediction of BoT completion.

SpeQuloS takes advantage of hybrid and elastic DCIs by dynamically allocating reliable resources from Clouds to compensate volatility of BE-DCIs nodes. The issue of outliers' tasks slowing down parallel executions is known for MapReduce applications and has been addressed by systems such as Mantri [3]. We propose a different approach which does not require knowledge of the resources that make up the infrastructure. By monitoring the BoT execution progress, very few information are needed to detect the tail effect. This allows delivering SpeQuloS as an on-line multi-BoT, multi-users service and able to serve several BE-DCI simultaneously. In this article, we investigate several strategies to decide when to assign tasks to Cloud workers and how to provision Cloud resources. These strategies are based on various metrics such as BoT completion thresholds and task execution variance. Strategies are evaluated and compared using a trace-driven simulator based from existing Grid, Cloud, and Desktop Grid infrastructures. Our simulator models two middleware which represents two different approaches for handling hosts volatility: BOINC, which relies on task deadlines and task replication and XtremWeb-HEP (XWHEP), which implements host failure detector based on heartbeats.

Performance evaluation results show that SpeQuloS is able to effectively remove the tail effect that delays BoT completion. In half of executions, the tail is totally removed and is significantly reduced in the other half. As a consequence, both for XtremWeb-HEP and BOINC the execution of BoT applications is greatly improved on every BE-DCIs investigated, and for any kinds of BoT workloads: An execution speed-up greater than 2 can be achieved. In addition, Cloud provisioning strategies implemented are able to minimize the usage of the Cloud resources: On average, less than 2.5 % of the BoT workload needs to be offloaded on Cloud resources. Finally, our evaluation shows that SpeQuloS can provide an accurate estimation of BoT completion time in 90 % of cases, greatly improving hybrid DCI user experience.

We also report on the SpeQuloS framework and its implementation. SpeQuloS has been designed to be deployed in complex hybrid infrastructures involving multiple platforms and spanning across several administration domains. It supports various Desktop Grid middleware (XtremWeb-HEP, BOINC) and Cloud technologies (Amazon EC2, OpenNebula, Nimbus, Rackspace). Its architecture is modular and distributed through several independent components.

We present various scenarios involving hybrid infrastructures and SpeQuloS: SpeQuloS is deployed on part of the production European Desktop Grid Infrastructure (EDGI), which interconnects several private and public Desktop Grids, Grids and private Clouds across all Europe. In this context, SpeQuloS is employed to improve QoS delivered to Desktop Grid users, using resources from EDGI private Clouds. Another use-case is presented where SpeQuloS lowers the cost of BoT execution on Amazon EC2. To achieve this, Spot Instances are provisioned to execute the major part of the BoT and when necessary, SpeQuloS instantiates regular instances to keep a satisfactory execution time. Finally, SpeQuloS can be deployed on Grids, allowing harvesting unused resources of these platforms while maintaining a high QoS level by supplying stable resources when needed. We present such a scenario in Grid5000.

The rest of the paper is organized as follow. In Sect. 2, we introduce our analysis of running BoT applications on best effort infrastructures. The SpeQuloS framework is presented in Sect. 3. Section 4 presents performance evaluation.

Section 5 reports on use-cases. Related works are presented in Sect. 6, and finally we conclude in Sect. 7.

2 Best effort distributed computing infrastructures

In this section, we define Best Effort Distributed Computing Infrastructures (BE-DCIs). The key principle of BE-DCIs is that participating nodes can leave the computation at any moment. We investigate how this characteristic impacts on BoT execution performance.

2.1 BE-DCI types

The different types of BE-DCIs that we study are as follows:

Desktop Grids (DGs) are grids composed of regular desktop computers typically used for computation when no user activity is detected. A node becomes unavailable when the user resumes his activity or when the computer is turned off. DGs can be supported by volunteer computing projects, such as SETI@home, where individuals offer their computing resources. DGs can also be internal to an institution which uses its collection of desktop computers to build a computational Grid.

Best Effort Grids are regular Grids used in Best Effort mode. Grid resource management systems, such as OAR [11], allow submission in a Best Effort queue. Tasks submitted to that queue have a lower priority and can be preempted by any other tasks. Therefore, if available grid resources are exhausted when a regular task is submitted, the resource manager kills as many best effort tasks as needed to allow its execution.

Cloud Spot Instances are variable-priced instances provided by Amazon EC2 Cloud service. Contrary to regular EC2 instances, which have a fixed price per hour of utilization, Spot instance prices vary according to a *market* price. A user can *bid* for a Spot instance by declaring how much he is willing to pay for one hour of utilization. If the market price goes lower than the user's bid, the instance is started. The user will only be charged at the price of the market, not at its bid price. If the market price goes higher than the bid, the instance is stopped. The Nimbus Cloud system has recently added support for Cloud Spot instances, as well as "Backfill" instances [30], which are low priority instances started when host resources are unused.

2.2 BoT execution on BE-DCIs

Bag of Tasks (BoT) are set of tasks that can be executed individually. Although there are many solutions for BoT execution on cross-infrastructure deployments, we assume that a Desktop Grid middleware is used to schedule tasks on the computing resources. We adopt the following terminology to describe the main components of Desktop Grid middleware: the server which schedules tasks, the user who submits tasks to the server, and workers which fetch and execute tasks on the computing resources.

Desktop Grid middleware have several desired features to manage BE-DCI resources: resilience to node failures, no reconfiguration when new nodes are added, task replication or task rescheduling in case of node failures and push/pull protocols that help with firewall issues. We consider two well established Desktop Grid middleware: BOINC which runs many large popular volunteer computing projects such as SETI@Home, and XtremWeb-HEP, which is an evolution of XtremWeb for the EGI Grid and implements several security improvements such as handling of Grid certificates. Condor and OurGrid would have also been excellent candidates, but we focus on middleware already deployed in EDGI infrastructure.

User tasks are submitted to the BOINC or XtremWeb-HEP server. Then, depending on the BE-DCIs targeted, the BoT is executed in the following way:

- On Desktop Grids, a desktop node runs the worker software.
- On the Grid, the worker software is submitted as a PilotJob, i.e. when the Grid task is executed, it starts the worker, which connects to the DG server and can start executing tasks from this server.
- When using Cloud resources, we follow a similar procedure by creating an instance, which contains the worker software and runs it at start-up.

Several projects [17, 28, 36] follow a similar approach, and find it to be efficient and scalable.

We captured several BoT executions, using the experimental environment described in Sect. 4.1. BoT execution profiles denote a slowdown in BoT completion rate during the last part of its execution. Indeed, examination of individual BoT execution traces showed that most of time, BoTs execution progression follows a pattern illustrated by Fig. 1: The last fraction of the BoT takes a large part of the total execution time. We called this phenomenon the **tail effect**. Many factors can be responsible of tail effect, such as failure and volatility of computing nodes, tasks heterogeneity and inefficient scheduling. Causes of this problem are not discussed here.

To characterize this tail effect, we investigate the difference between the BoT actual completion time and an ideal completion time. The ideal completion time is the BoT completion time that would be achieved if the completion rate, calculated at 90 % of the BoT completion, was constant. Therefore, the ideal completion time is $\frac{t_c(0.9)}{0.9}$, where $t_c(0.9)$ is the elapsed time when 90 % of the BoT is completed. Figure 1 illustrates this definition. The ideal completion time is computed at 90 % of completion because we observed that

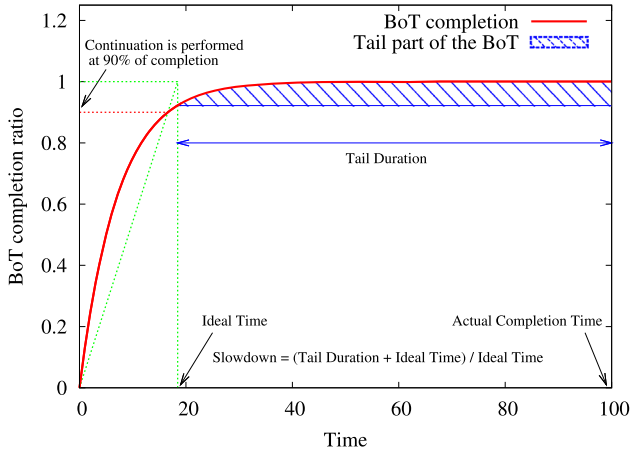


Fig. 1 Example of BoT execution with noteworthy values

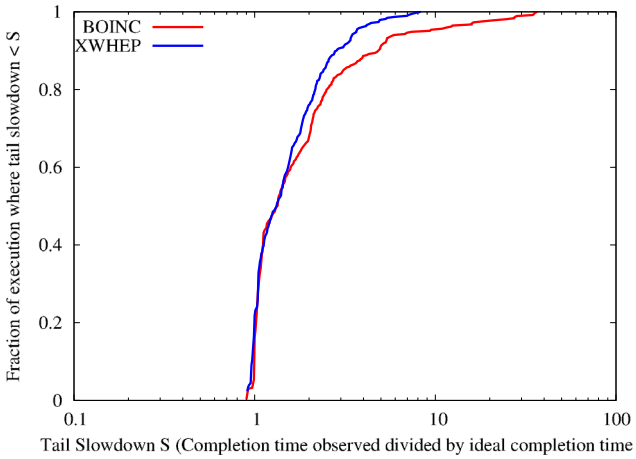


Fig. 2 Profiling execution of BoTs in BE-DCIs: Tail Slowdown is the BoT completion time divided by the ideal completion time (i.e. determined by assuming a constant completion rate). The cumulative distribution function of observed slowdowns is represented

except during start-up, the BoT completion rate remains approximately constant up to this stage of execution. Therefore, the ideal completion would have been equivalent if it had been calculated at 50 % or 75 % of BoT completion.

Intuitively, the ideal completion time could be obtained in an infrastructure which would offer constant computing capabilities.

We define the *tail slowdown* metric as the ratio between ideal completion time and actual BoT completion time. The tail slowdown reflects the BoT completion time increase factor resulting from the tail effect. Figure 2 presents the cumulative distribution functions of tail slowdowns observed during BoT executions in various BE-DCI environments.

One can observe that the distribution is largely skewed and in some cases, the slowdown seriously impacts BoT completion time. About one half of BoT executions are not extremely affected by the tail effect. In those cases, the tail slowdown does not exceed 1.33, meaning that the tail effect slows the execution by no more than 33 %. Other cases are less favorable; the tail effect doubles the completion time from 25 % of executions with XWHEP middleware to 33 % with BOINC. In the worst 5 % of execution, the tail slowdown ranges from 400 % with XWHEP to 1000 % for BOINC. These results are mostly due to host volatility and the fact that Desktop Grid middleware have to wait for failure detection before reassigning tasks.

Table 1 Average fraction of Bag of Tasks in the tail, i.e. the ratio between the number of tasks in the tail versus the total number of tasks in the BoT and average percentage of execution time in tail, i.e. the percentage of BoT execution time (makespan) spent in the tail

BE-DCI Trace	Avg. % of BoT in tail		Avg. % of time in tail	
	BOINC	XWHEP	BOINC	XWHEP
Desktop Grids	4.65	5.11	51.8	45.2
Best Effort Grids	3.74	6.40	27.4	16.5
Spot Instances	2.94	5.19	22.7	21.6

The tail part of a BoT execution is the set of tasks executed during the tail effect, i.e. later than the ideal completion time. These tasks create the tail effect by taking unusually long to complete. Table 1 shows characteristics of BoT tails, according to middleware and types of BE-DCIs considered.

In the table, we see that a few percent of BoTs' tasks belong to the tail, whereas a significant part of the execution takes place during the tail. Therefore, the completion time of a small fraction of a BoT is many times longer than completion time of most of the BoT. This also explains why the ideal time remains approximately the same when it is calculated up to 90 % of BoT completion; the tail effect never appears before that stage.

Results of this section show that the tail effect can affect all kind of BE-DCIs, whatever is its volatility, or amount of resources, for both BOINC and XWHEP middleware. It may strongly slow down the completion time of BoTs executed on BE-DCIs and cause high execution variance, precluding any performance prediction.

3 SpeQuloS

In this section, we are describing SpeQuloS service and implementation, which aims at providing QoS to BoT execution on BE-DCIs.

3.1 Overview of the SpeQuloS service

SpeQuloS is a service which provides QoS to users of Best Effort DCIs by provisioning stable resources from Cloud services.

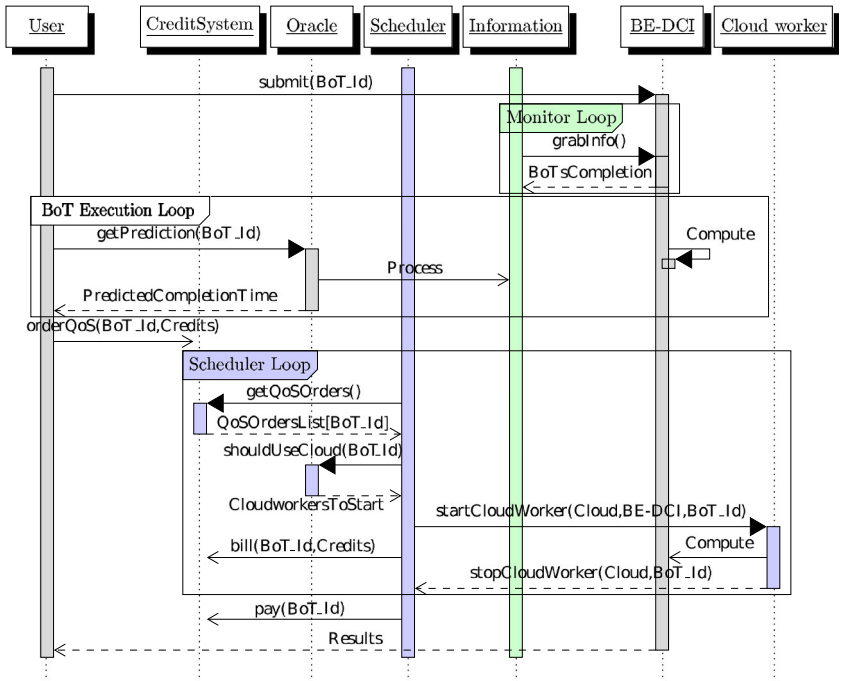


Fig. 3 Sequence diagram of SpeQuloS interactions in a typical use-case scenario

To supply resources to a BE-DCI, SpeQuloS uses Infrastructure as a Service (IaaS) Cloud to instantiate a virtual instance, called a Cloud worker. To be able to process tasks from the BE-DCI, a Cloud worker typically runs the DG middleware worker software that is used in the BE-DCI.

SpeQuloS implements various strategies to ensure efficient usage of Cloud resources and provides QoS features to BE-DCI users. As access to Cloud resources is costly, SpeQuloS provides a framework to regulate access to those resources among users and account for their utilization.

SpeQuloS is composed of several modules as shown in Fig. 3. The Information module gathers and stores information from BE-DCIs (see Sect. 3.2). The Credit System module is in charge of the billing and accounting of Cloud-related operations (Sect. 3.3). The Oracle module helps SpeQuloS determine how to efficiently deploy the Cloud resources, and gives QoS information to users (Sects. 3.4 and 3.5). The Scheduler module manages the BoT and the Cloud workers during its execution (Sect. 3.6).

Figure 3 presents a simplified sequence diagram of a typical usage of SpeQuloS and the different interactions between the components of the system.

The progression of the scenario is represented vertically, and the various function calls between SpeQuloS modules are represented by arrows. A description of the various steps of this scenario is as follows:

- The first step of the scenario is a user submitting a BoT tagged with a unique identifier (`BoT_Id`). The BoT execution is then monitored by the Information module.
- At any moment, the user can request the Oracle to predict the BoT completion time to estimate QoS benefits of using Cloud resources. Then, the user may order to the Credit System QoS support for his BoT by allocating an amount of credits. The Credit System verifies that there are enough credits on the user's account to allow the order, and then it provisions credits to the BoT.
- The Scheduler periodically asks the Credit System if there are credits allocated for some BoTs. If credits are provisioned for a BoT, it asks the Oracle if it should start Cloud workers to accelerate the BoT execution.
- Cloud workers are started by the Scheduler to take part in the BoT execution. The Scheduler has to ensure that appropriate BE-DCI tasks are assigned to Cloud workers.
- At each fixed period of time, the Cloud resource usage must be billed. For each Cloud worker started, the Scheduler reports to the Credit System the corresponding credits used. If all the credits allocated to the BoT have been spent, or if the BoT execution is completed, Cloud workers are stopped.
- The Scheduler finally asks the Credit System to pay for the Cloud resources usage. The Credit System closes the order relative to the BoT. If the BoT execution was completed before all the credits have been spent, the Credit System transfers back the remaining credits to the user's account.

3.2 Monitoring BoT executions

SpeQuloS collects information on BoT executions which are relevant to implement QoS strategies with two objectives: (1) provide real-time information on BoT execution and BE-DCI computational activities and (2) archive BoT execution

traces from which a statistical model can be extracted in order to compute a prediction of BoT execution time. To do so, the Information module stores in a database the BoT completion history as a time series of the number of completed tasks, the number of tasks assigned to workers and the number of tasks waiting in the scheduler queue. The amount of information transmitted per BoT is less than few hundreds bytes per minute, which allows the system to handle many BoTs and infrastructures simultaneously.

One key point is to hide infrastructure idiosyncrasies, i.e., different Desktop Grid middleware that have specific ways of managing queues should appear in a unified format. Because we monitor the BoT execution progress, a single QoS mechanism can be applied to a variety of different infrastructures.

3.3 Cloud usage accounting and arbitration

Because Cloud resources are costly and shared among users, a mechanism is required to account for Cloud resource usage and to enable Cloud usage arbitration. The Credit System module provides a simple credit system whose interface is similar to banking. It allows depositing, billing and paying via virtual credits.

BE-DCI users spend their credits to support a BoT execution. Credits denote an amount of Cloud worker usage. At the moment, the Credit Systems uses a fixed exchange rate; 1 *CPU.hour* of Cloud worker usage costs 15 credits. At the end of the BoT execution, the amount of credits corresponding to the actual usage of Cloud resources is withdrawn from the user's credit account.

SpeQuloS manages users' accounts. A deposit policy is used by administrators for the provisioning of these accounts. Although simple, the system is flexible enough to give administrators control over Cloud usage. For instance, a simple policy that limits SpeQuloS usage of a Cloud to 200 nodes per day would be to write a deposit function, run once every 24 hours, which deposits $d = \max(200 \times node\_cost\_per\_hour \times 24, 200 \times node\_cost\_per\_hour \times 24 - user\_credit\_spent)$ credits into an account. Furthermore, the mechanism allows one to easily implement more complex policies, such as the "network of favors" [5], which would allow cooperation among multiple BE-DCIs and multiple Clouds providers.

3.4 Providing QoS estimation to BE-DCI users

Providing QoS features to BE-DCI users requires one to appropriately inform these users on the QoS level they can expect. These objectives are the responsibility of the Oracle module and are allowed by a careful exploitation of history of BoT execution traces collected by the Information module as well as real-time information about the progress of BoT execution. With this information, the Oracle module is able to compute a predicted completion time for the BoT. This prediction helps users to decide if it worth spending credits for BoT QoS.

The following prediction methods are currently used in SpeQuloS: when a user asks for a prediction, SpeQuloS retrieves the current user BoT completion ratio (r) and the elapsed time since BoT submission ($t_c(r)$), using the BoTs execution history stored in the Information module. It computes the predicted completion time t_p as: $t_p = \alpha \frac{t_c(r)}{r}$. SpeQuloS then returns this predicted time and its associated statistical uncertainty.

The α factor allows one to adjust the prediction based on the history of previous BoT executions in a given BE-DCI. At initialization, α factor is set to 1. Then, after some BoTs executions, the value of α is adjusted to minimize the average difference between the predicted time and the completion times actually observed. The statistical uncertainty returned to the user is the success rate (with a ± 20 % tolerance) of predictions performed on previous BoT executions, observed from the historical data.

3.5 Cloud resources provisioning strategies

We design and evaluate several different strategies for the Oracle module to decide when and how many Cloud workers should be started. We introduce three strategies to decide when to launch Cloud workers:

- Completion Threshold (9C): Cloud workers are started as soon as the number of completed tasks reaches 90 % of the total BoT size.
- Assignment Threshold (9A): Cloud workers are started as soon as the number of tasks assigned to workers reaches 90 % of total BoT size.
- Execution Variance (V): Let $t_c(x)$ be the time at which x percent of BoT tasks are completed and $t_a(x)$ be the time at which x percent of BoT tasks were assigned to workers. We call the execution variance $\text{var}(x) = t_c(x) - t_a(x)$. Intuitively, the sudden change in the execution variance indicates that the system in no longer in steady state. Cloud workers are launched when the execution variance doubles compared to the maximum one measured during the first half of the BoT execution. More precisely, if c is the fraction of the BoT completed, Cloud workers are started as soon as:

$$\text{var}(c) \geq 2 \max_{x \in [0,50\ \%]} \big(\text{var}(x)\big)$$

Assuming that users spend an amount of credits corresponding to S CPU.hours of Cloud usage, we propose two approaches to decide how many Cloud workers to start:

- Greedy (G): S workers are immediately started. Cloud workers that do not have tasks assigned stop immediately

to release the credits. Doing so, other workers which have obtained tasks can complete their task.

– Conservative (C): Let $t_c(x)$ be the elapsed time at which x percent of BoT tasks are completed. Then $\frac{t_c(x)}{x}$ is the BoT completion rate. At time t_e, x_e and $t_c(x_e)$ are known from the SpeQuloS Information module monitoring. We can give an estimation of the remaining time t_r by assuming a constant completion rate:

$$t_r = t_c(1) - t_e = t_c(1) - t_c(x_e) = \frac{t_c(x_t)}{x_t} - t_c(x_t)$$

Then, $\max(\frac{S}{t_r}, S)$ Cloud workers are launched, ensuring that there will be enough credits for them to run during the estimated time needed for the BoT to complete.

We present three methods in the way of using these Cloud resources:

– Flat (F): Cloud workers are not differentiated from any regular workers by the DG server. Thus, in this strategy, all workers compete to get the remaining tasks of the tail.
– Reschedule (R): In contrast with Flat, the DG server differentiates Cloud workers from the regular workers. Cloud workers are served first with pending tasks if there are some, and if not with a duplicate of the tasks which are being executed on regular workers. This strategy ensures that tasks executed on regular workers and which may cause the tail are scheduled in the Cloud. However, the strategy is optimistic in the sense that it allows a regular worker which has computed a result to send the result and finish the task.
– Cloud Duplication (D): Cloud workers do not connect to DG server, but connect to a dedicated server hosted in the Cloud. All uncompleted tasks (even those under execution) are duplicated from the DG server to this Cloud server and are processed by Cloud workers. This strategy allows one to execute all the tasks of the tail on the stable Cloud resources, while keeping Cloud workers separated from regular Cloud workers.

Note that these strategies have different implementation complexities. Flat is the simplest one which does not need modification of the DG scheduler. Reschedule requires one to modify the DG scheduler in order to differentiate Cloud workers from regular one, which is not always possible in a production infrastructure where system administrators are reluctant to patch their DG servers. Cloud Duplication allows one to keep the DG scheduler unchanged, and therefore is transparent to the BE-DCI. But requires that SpeQuloS implement the task duplication from DG to Cloud server and the merging of results coming from Cloud workers and the regular BE-DCI.

3.6 Starting workers on the cloud

The Scheduler module manages the Cloud resources provisioned to support execution of the BoT for which users have required QoS. If credits have been allocated, and the Oracle decides that Cloud workers are needed, the Scheduler starts Cloud workers to support a BoT execution. As soon as Cloud resources are not needed anymore, or allocated credits are exhausted, the Cloud workers are shutdown remotely.

Algorithm 1 MONITORING BOT

```
for all B in BoTs do
  if Oracle.shouldUseCloud(B) then
    if CreditSystem.hasCredits(B) then
      for all CW in Oracle.cloudWorkersToStart(B) do
        CW.start()
        configure(B.getDCI(),CW)
      end for
    end if
  end if
end for
```

Algorithm 2 MONITORING CLOUD WORKERS

```
for all CW in startedCloudWorkers do
  B ← CW.getSupportedBoT()
  if (Info.isCompleted(B)) or
  (not CreditSystem.hasCredits(B)) then
    CW.stop()
  else
    CreditSystem.bill(B,CW)
  end if
end for
```

Technically, this feature is achieved by building Cloud instances which embed the DG worker middleware. We use the *libcloud* library, which allows unifying access to various IaaS Cloud technologies in a single API. Once the Cloud worker is executed on a Cloud resource, the Scheduler connects through SSH to the instance and configures the worker to connect to the BE-DCI for processing tasks from the appropriate BoT. Indeed, it is important to ensure that a Cloud worker on which a user is spending credits is not computing tasks belonging to other users.

Algorithms 1 and 2 present the various operations performed by the Scheduler module to monitor BoT execution and to manage Cloud workers.

3.7 Implementation

SpeQuloS has been developed as a set of independent modules, using the Python programming language and MySQL databases. Communication between modules use web services, therefore modules can be deployed on different networks. Several BE-DCIs and Cloud services can be connected at the same time to a single SpeQuloS server.

The SpeQuloS implementation targets a production level of quality. Testing and deployment are performed by differ-

ent teams of the EDGI consortium. The SpeQuloS source code is publicly available.[1]

3.7.1 Desktop Grid's middleware and Grid's integration

SpeQuloS supports both BOINC and XWHEP middleware which are used in BE-DCIs. To distinguish QoS-enabled BoT from others, tasks belonging to these BoT are tagged by the users using a special field in the middleware task description (`batchid` in BOINC and `xwgroup` in XWHEP).

One issue is to ensure that Cloud workers only compute tasks belonging to the BoT for which credits has been provisioned. We solve this situation in BOINC by adding a new policy to the matchmaking mechanism. Note that BOINC requires that scheduling policies be coded and specified by compile time, which requires patching the BOINC server. For XWHEP, developers agreed to include a new configuration option in version 7.4.0 that met our needs.

Another challenge is to enable SpeQuloS support in hybrid infrastructures, where regular Grids are used. The 3G-Bridge [37] developed by SZTAKI is used in the EDGI infrastructure to provide Grid and Desktop Grid interoperability. Tasks submitted to a regular Grid computing element connected to the 3G-Bridge may be transparently redirected to a Desktop Grid. To enable SpeQuloS support of BoTs submitted using the 3G-Bridge, it has been adapted to store the identifier used by SpeQuloS to recognize a QoS-enabled BoT.

3.7.2 Cloud services support

Thanks to the versatility of the libcloud library, SpeQuloS supports the following IaaS Cloud technologies: Amazon EC2 and Eucalyptus (which are two compliant technologies deployed either on commercial or private Clouds), Rackspace (which is a commercial Cloud), OpenNebula and StratusLab (which implements the Open Cloud Computing Interface specification, delivered through the Open Grid Forum), and Nimbus (a Cloud system targeting scientists). In addition, we have developed a new driver for libcloud so that SpeQuloS can use Grid5000 [8] as an IaaS cloud.

4 Evaluation

In this section we report on the performance evaluation of SpeQuloS using simulations.

We have developed simulator of BOINC and XWHEP, which uses node availability traces from real infrastructure and generates traces of BoT execution. It also optionally simulates SpeQuloS utilization.

[1] http://graal.ens-lyon.fr/~sdelamar/spequlos/.

4.1 Simulations setup

4.1.1 BE-DCIs availability traces

There have been many studies around nodes volatility for BE-DCIs. In particular several data-sets are provided by the Failure Trace Archive [24]. However, to our knowledge, there was no availability measurement for Cloud Spot instances or Grid systems used in best effort mode. We collected the following traces:

- *Desktop Grid:* For this study we consider the public volunteer computing project SETI@Home (`seti`) ran by BOINC [22], and the private Desktop Grid deployments at University Notre Dame, ran by Condor [35] (`nd`). All these traces are provided by the Failure Trace Archive [24].
- *Best Effort Grid:* We consider the best effort queues of Grid5000 [8] (G5K) infrastructure. We generated traces from the Gantt utilization charts for both Lyon (`g5klyo`) and Grenoble (`g5kgre`) G5K clusters for December 2010 period. The unused resources reported in the charts are considered as resources available in best effort. In other words, a node is available in Best Effort Grid traces when it does not compute regular tasks, and vice-versa.
- *Cloud Spot instances:* Cloud Spot instances such as Amazon EC2 Spot instances are variable-priced instances. These instances are only started if a user bid is higher than their current price. Thus, with Spot instances, the host availability depends both on the user's bids and the instance price market variation.

  We consider the following usage of Spot instance: a total renting cost per hour (S) is set by the user to use several instances. As this cost is constant while the market price varies, the number of provisioned instances will vary. To implement this scenario, we use the following strategy: We place a sequence of n bids at price $\frac{S}{i}$, where $i \in 1..n$. n should be chosen high enough so that $\frac{S}{n}$ is lower than the lowest possible Spot Instance price. Hence, we ensure that the maximum number of Spot Instances is started for total renting cost of S.

  Bids are placed using the *persistent* feature, which ensures that the requests will remain in consideration after each instance termination. Using price market history provided by Amazon from January to March 2011, we have generated the instances availability traces of the *c1.large* instance for a renting cost of 10 dollars (`spot10`) and 100 dollars (`spot100`) per hour.

Variation of the number of nodes over time Figure 4 and Table 2 present the number of available nodes during the first 10 days of each trace. The figure emphasizes the diversity among infrastructures. *seti* contains more than 10000 nodes whereas other infrastructures have between 100 and

1000 nodes. Number of nodes variability also differs. It is high for some traces (*g5klyon*), and less important for others (*spot10*, *nd*). Some periodic patterns can also be observed in some traces. Indeed, each 24 hours, repetitive patterns are exhibited in *seti* traces: we can see that some nodes are turned off during nights and weekends.

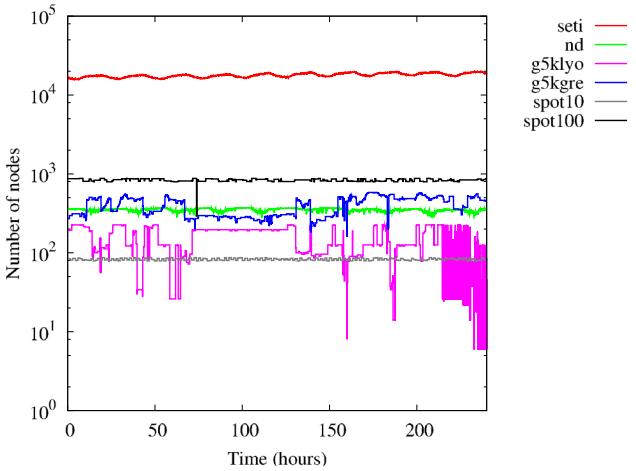


Fig. 4 Number of nodes during the first 10 days of BE traces

Availability and unavailability periods Figures 5(a), 5(b), summarized in Table 3, show distributions of availability and unavailability periods of nodes for each trace. Availability periods are various uninterrupted periods when a node can participate to computation. Unavailability periods are periods comprised between two successive availability periods.

Figure 5(a) shows differences between traces. In some infrastructures, the node volatility is high: nodes are likely to stay available only during few minutes, or even seconds (*g5klyo*). Volatility may be low: nodes usually stay available for more than tens of minutes (*nd*, *spot10*, *spot100*). Finally, some BE-DCIs exhibit a "mixed" volatility pattern (*g5kgre*, *seti*): Some availability periods are short whereas others are longer.

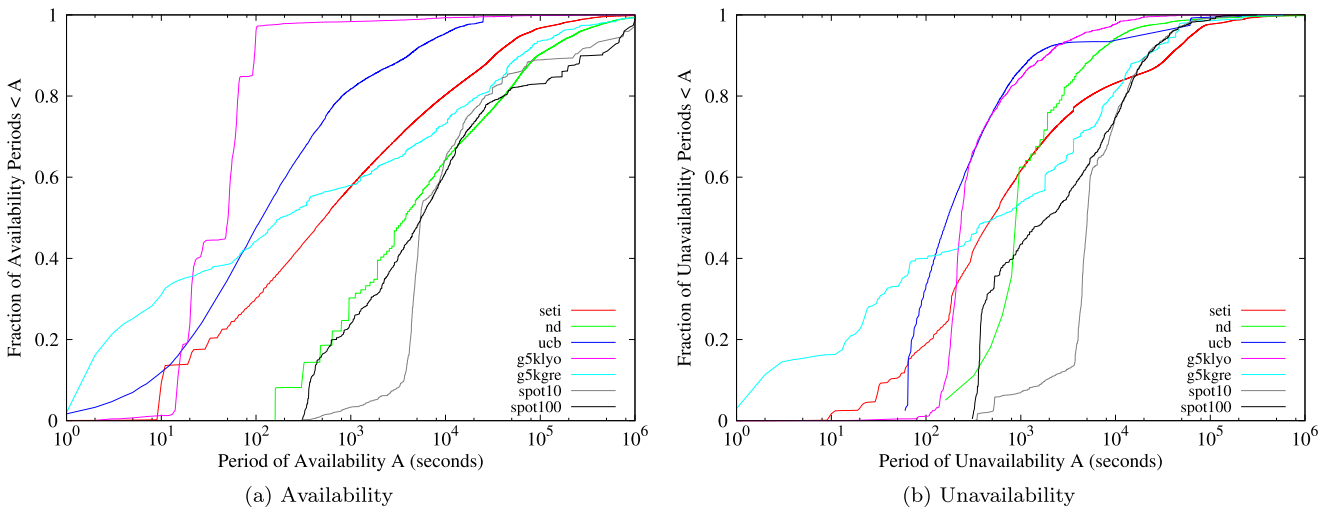
In Fig. 5(b), fewer differences are observed. Indeed, most of unavailability periods are comprised between one minute and one hour. However, highly volatile infrastructures tend

Table 2 Number of nodes in Best Effort DCI traces. The trace length, number of nodes average (*Mean*), standard deviation (*Deviation*), minimum (*Min*) and maximum (*Max*) are presented

Trace	Length (days)	Mean	Deviation	Min	Max
seti	120	24391	6793	15868	31092
nd	413.87	180	4.129	77	501
g5klyo	31	90.573	105.4	6	226
g5kgre	31	474.69	178.7	184	591
spot10	90	82.186	3.814	29	87
spot100	90	823.95	4.945	196	877

Table 3 Availability and unavailability of Best Effort DCI nodes. *Av. quartiles* and *Unav. quartiles* are the nodes availability and unavailability duration quartiles, in seconds

Trace	Av. quartiles (s)	Unav. quartiles (s)
seti	61, 531, 5407	174, 501, 3078
nd	952, 3840, 26562	640, 960, 1920
g5klyo	21, 51, 63	191, 236, 480
g5kgre	5, 182, 11268	23, 547, 6891
spot10	4415, 5432, 17109	4162, 5034, 9976
spot100	1063, 5566, 22490	383, 1906, 10274



(a) Availability (b) Unavailability

Fig. 5 Nodes availability and unavailability periods cumulative distribution functions

to have shorter unavailability periods, whereas low volatile infrastructures have longer ones, and mixed volatile infrastructures have mixed unavailability periods.

Computing power Computing power of BE-DCI nodes depends on its nature. As DG workers use regular desktop computers, their computing power is much lower than Grid or Cloud ones. In addition, whereas Grid computing resources are usually homogeneous, DG and even Cloud resources show heterogeneity. Previous works [18, 25] allow us to model nodes power. Table 4 shows BE-DCIs workers computing power drawn from those studies: Cloud and Grid nodes are three times faster than DG nodes average and DG and Cloud computing power is heterogeneous and follows a normal distribution.

Summary The BE-DCI availability traces presented demonstrate diversity among infrastructures. However, these traces can be classified along common attributes, such as the amount of resources, volatility and presence of cycles.

Table 4 Computing power of Best Effort DCI nodes. *Avg. power* and *Power std. dev.* are the average node power (in instructions per second) and node power standard deviation

Trace	Avg. power (nops/s)	Power std. dev.
seti	1000	250
nd	1000	250
g5klyo	3000	0
g5kgre	3000	0
spot10	3000	300
spot100	3000	300

Table 5 summarizes the characteristics of the BE-DCI investigated in this document.

4.1.2 BoT workloads

BoT applications are a major source of DCIs workload. We follow the definition of BoT given in [20, 31] where a BoT is an ordered set of n independent tasks: $\beta = \{T_1, \ldots, T_n\}$. All tasks in β have the same owner and the same group name or group identifier. In addition, Desktop Grid systems impose users to register applications in the server, thus we also have the requirement that tasks refer to the same application.

Tasks may not be submitted at the same time. We define $AT(T_i)$, the arrival time of the task T_i and we have $AT(T_i) < AT(T_j)$ if $i < j$. More, we define ϵ, the maximal time between two tasks arrivals, thus we have $\forall i \in (1, \ldots, n), AT(T_{i+1}) - AT(T_i) < \epsilon$. A typical ϵ value is 60 seconds, as used in [31].

BoTs are also defined by their *size* i.e. the number of tasks. Each task also has a number *nops* of instructions to be processed. In homogeneous BoT, all the tasks have the same number of instructions. Conversely, in heterogeneous BoTs, the number of operations per tasks follows a probabilistic distribution.

The BoT workloads that we selected in our experimentation come from our experience in distributed computing infrastructures, such as the ones used in the EDGI project. The BIG workload is representative of BoT observed in public volunteer computing projects, and SMALL workload is representative of BoT observed in Grids such as Grid5000 [19]. The RANDOM workload is statistically generated based on scientific studies conducted by Minh and Al, cited in [31]. Those BoTs vary in terms of size, number of instructions per task and task arrival times. Table 6 summarizes the BoT

Table 5 Main attributes of investigated Best Effort DCIs

Trace	Amount of nodes	Cycles	Volatility	Computing power
g5klyo	hundreds	no	high	high
g5kgre	hundreds	no	mixed	high
seti	thousands	yes	mixed	low
nd	hundreds	no	low	low
spot10	hundreds	no	low	high
spot100	hundreds	no	low	high

Table 6 Characteristic of BoT workload: *size* is the number of tasks in the BoT, *nops/task* is the number of instructions per tasks and *arrival* the repartition function of tasks arrival time. *weib* is the Weibull distribution and *norm*, the Normal distribution

	Size	nops/task	Arrival time
SMALL	1000	3600000	0
BIG	10000	60000	0
RANDOM	norm($\mu = 1000, \sigma^2 = 200$)	norm($\mu = 60000, \sigma^2 = 10000$)	weib($\lambda = 91.98, k = 0.57$)

attributes. As shown in the table, SMALL and LARGE BoTs are homogeneous BoT, whereas RANDOM is heterogeneous.

4.1.3 Simulations parameters

Simulators are configured with DG middleware standard parameters. For the BOINC simulator, each task is replicated 3 times (*target_nresult* = 3), and 2 replicas results are needed to consider a task completed (*min_quorum* = 2). Two task replicas cannot be executed on the same worker (*one_result_per_user_per_wu* = 1). After it is assigned to a worker, the maximum time to receive a replica result before reassigning it is set to 1 day (*delay_bound* = 86400). For XW simulator, workers send a *keep alive* message every minute (*keep_alive_period* = 60). When the server does not receive any *keep alive* message from a worker for 15 minutes (*worker_timeout* = 900), it reassigns task executed on this worker to another one.

Pseudorandom number generator used in simulators can be initialized by a seed value to reproduce exactly the same simulation executions. Therefore, using the same seed value allows a fair comparison between a BoT execution where SpeQuloS is used and the same execution without SpeQuloS.

SpeQuloS users can choose the amount of credits they allocate to support BoT executions. In simulations, the amount of credits is set to be equivalent, in terms of *CPU.hour*, to 10 % of total BoT workload. Therefore, depending on the BoT category considered, the number of provisioned credits varies. The BoT workload is computed as its size multiplied by tasks' wall clock time. Task wall clock time is an estimated upper bound for individual task execution time and is set to 11000 seconds for SMALL BoTs, 180 seconds for BIG BoTs and 2200 seconds for RANDOM BoTs.

The simulator executes the various BoTs described in Table 6 on selected BE-DCIs representative of Desktop Grids (seti, nd), Best Effort Grids (g5klyo, g5kgre) and Clouds (spot10, spot100), using BOINC and XWHEP. Different BoT submission times are used in order to simulate execution in different time period of the BE-DCI traces. Results of this section are produced thanks to simulations of more than 25000 BoT executions.

4.2 Evaluation of Cloud resources provisioning strategies

In this section, we report on the performance evaluation of SpeQuloS strategies for Cloud provisioning presented in Sect. 3.5. We evaluate every combinations of the strategies to find which one gives the best performance. We evaluate these combined strategies via trace-driven simulation for different middleware (BOINC or XWHEP), different BE-DCI availability traces, and different classes of BoTs. We look for the best strategy over all scenarios. The naming of the strategy combinations follows this scheme: 9A-G-D means that Cloud workers will start when 90 % of the tasks have been assigned (Assignment Threshold), all the Cloud workers are started at once (Greedy) and all uncompleted tasks are duplicated to the Cloud (Cloud Duplication).

4.2.1 Tail removal efficiency

The first experiment aims at comparing the efficiency of the Cloud provisioning strategies to alleviate the tail effect. We define the *Tail Removal Efficiency* (TRE) as the percentage reduction of the tail duration with SpeQuloS compared to without SpeQuloS. We calculate TRE as $\text{TRE} = 1 - \frac{t_{speq} - t_{ideal}}{t_{nospeq} - t_{ideal}}$, where $t_{nospeqs}$ is the completion time measured without SpeQuloS (which is likely to be affected by tail), t_{speq} is the completion time measured for the same BoT execution when SpeQuloS is used. t_{ideal} is the ideal completion time for that execution without the tail.

Figures 6(a), 6(b) and 6(c) present the complementary cumulative distribution function of TRE for several combinations of Cloud resource provisioning strategies. For a given efficiency, the figures show the fraction of BoT executions which obtained a greater efficiency.

We first observe that all the strategies are able to significantly address the tail effect. In the best cases (Fig. 6(c), 9A-G-D, 9A-C-D), the tail has disappeared in one half of the BoT executions (TRE = 100 %) and for 80 % of the BoT executions the tail has been at least halved (TRE > 50 %), which is satisfactory.

A comparison of the strategies shows that for the Flat deployment strategy (Fig. 6(a)) has the worst performances regardless of the combination used: in half of the BoT executions the tail has not been significantly reduced (TRE < 30 %). Reschedule (Fig. 6(b)) and Cloud Duplication strategies (Fig. 6(c)) both perform better than Flat if the Execution Variance is excluded: 80 % of the BoT executions have addressed the tail effect (TRE > 30 %). Clearly, the Execution Variance causes a severe drop of performance of any combinations which include this strategy. The Assignment threshold strategy has slightly better results than the Completion threshold strategy, and Reschedule is slightly better than Cloud duplication, especially when the Completion threshold strategy is used.

The Flat strategy cannot reach the same level of performance as the others because Cloud resources are in competition with BE-DCIs resources. In this strategy, tasks are assigned without distinction between Cloud workers and normal workers, which leads to Cloud workers not receiving tasks from DG server even during the tail part of the BoT execution. The Execution Variance strategy which tries to dynamically detect the tail effect by monitoring the variation of tasks' execution time, is shown to be less efficient

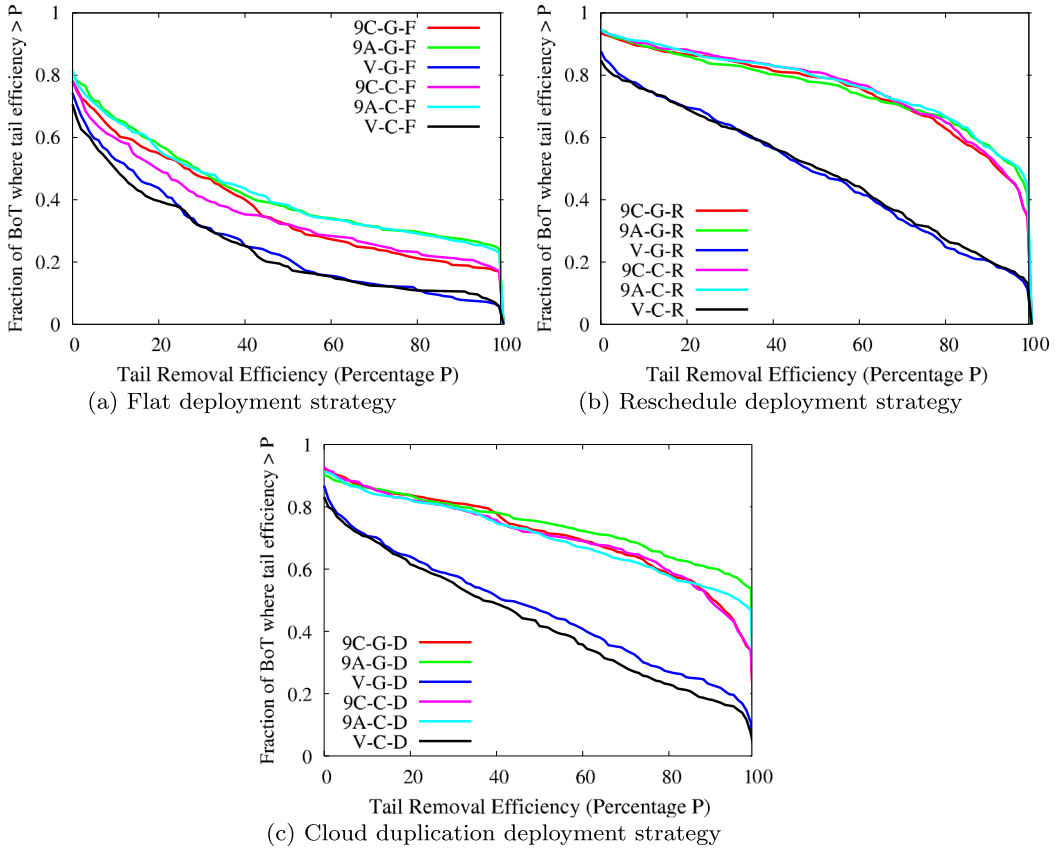


Fig. 6 Complementary cumulative distribution functions of Tail Removal Efficiency for several combinations of Cloud resources provisioning strategies. Tail removal efficiency denotes the reduction percentage of the tail duration using SpeQuloS compared to without SpeQuloS. Notation of strategies combination is described in Sect. 3.5

than the others. We observed that unfortunately this strategy starts Cloud workers too late for a significant number of executions.

4.2.2 *Cloud resource consumption*

The second criterion for the performance comparison of the strategies is the Cloud resource consumption. Lower is the resource consumption, better is the strategy. In our system, 1 CPU.hour of Cloud workers usage is billed as 15 credits. The metric used to measure the Cloud utilization is the number of credits spent during the execution.

Figure 7 shows the average percentage of credits spent against the credits provisioned. In most cases, less than 25 % of provisioned credits are spent. In our evaluation, provisioned credits are equivalent to 10 % of the total BoT workload in terms of Cloud worker *CPU.hours*. Our results mean that actually, less than 2.5 % of the BoT workload is executed in the Cloud, and so is the equivalent consumption of credits.

Figure 7 shows that credit consumption of the Cloud duplication strategy is lower than Flat which is lower than Reschedule. Indeed, in this last strategy, Cloud workers are

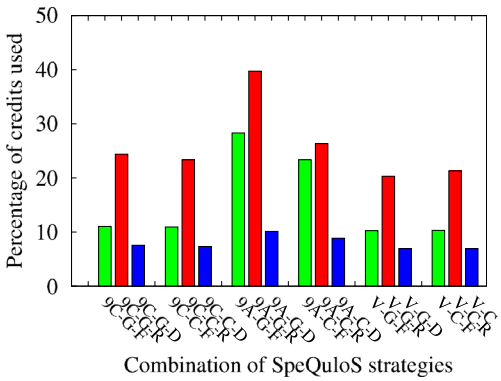


Fig. 7 Credits consumption of various SpeQuloS strategies combinations. Lower is better. Notation of strategies combination is described in Sect. 3.5

continuously busy because they receive uncompleted task duplicates until the BoT execution is finished. Results also show that Assignment threshold consumes more than the others because it starts Cloud workers earlier, and that Conservative method saves a little more credits than Greedy.

Overall, our strategies have low credit consumption. It ensures that enough credits are supplied to support the BoT

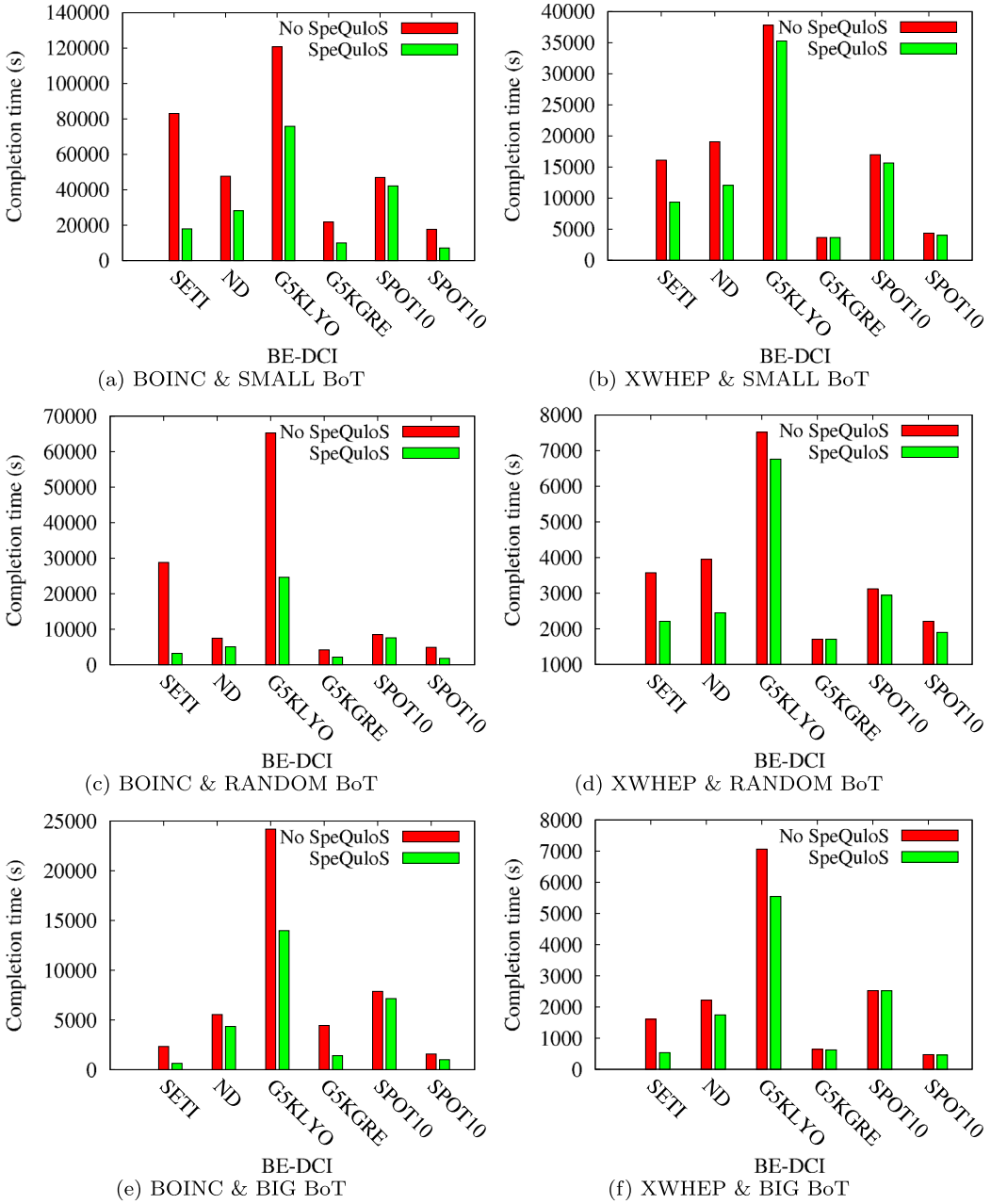


Fig. 8 Average completion time measured with and without SpeQuloS under various execution environments

execution until it ends and leaves more credits to users to support other BoT executions.

4.3 SpeQuloS performance

In this section, we evaluate SpeQuloS performance to effectively enhance QoS of BoT executed on BE-DCIs. The results of this section use the Completion threshold, Conservative and Reschedule (`9C-C-R`) strategy combination, which is a good compromise between Tail Removal Efficiency performance, credits consumption and ease of implementation.

4.3.1 Completion speedup

Figures 8(a), 8(e), 8(c), 8(b), 8(f) and 8(d) show the average BoT completion time measured with and without SpeQuloS. Each figure presents results from one DG middleware and BoT. Each figure pair of columns shows results for each BE-DCI trace.

The results show that in all cases, SpeQuloS decreases the completion time. Performance enhancement depends on the BE-DCI, BoT and middleware considered. More important gains are observed with BOINC, `seti`, and the

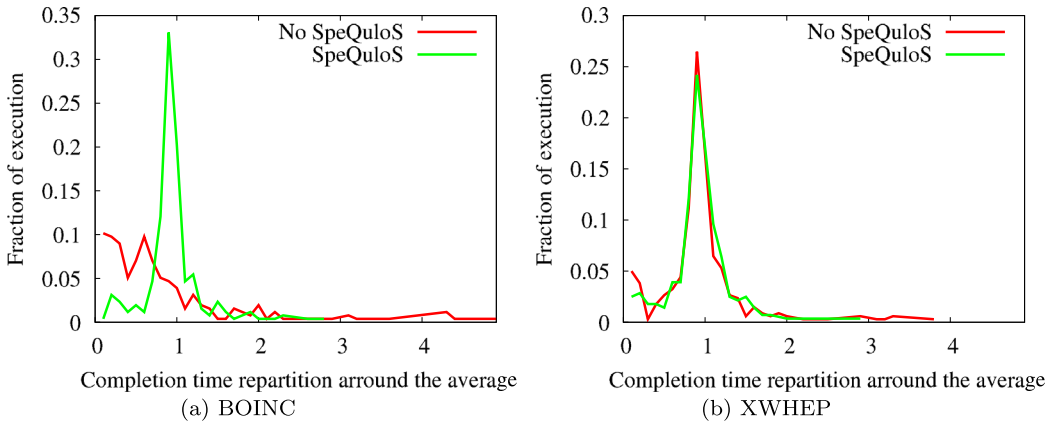


Fig. 9 Repartition functions of execution completion time normalized with the average completion time observed under same environment (BE-DCI traces, DG middleware, BoT). Curves centered around 1 denote stable executions

RANDOM BoT, for which average completion time is reduced from 28818 seconds to 3195 seconds. In contrast, with XWHEP, spot10 and BIG BoT, the average completion is not much improved (from 2524 to 2521 seconds).

More important benefits are observed with highly volatile BE-DCIs (seti, nd, g5klyo). As the tail effect is more important in these BE-DCIs, using SpeQuloS can significantly increase the performance.

Benefits are also more important for SMALL BoTs, which are made of long tasks, and RANDOM BoTs, which are heterogeneous, in particular with Desktop Grid DCIs (seti & nd), for which node characteristics (low power and high volatility) make it difficult to execute such BoTs without SpeQuloS.

Even if BOINC and XWHEP completion times cannot be compared, as these middleware differ in the way they detect and handle task execution failures, one can note that XWHEP is slightly less improved than BOINC when SpeQuloS is used.

4.3.2 *Execution stability*

One additional QoS enhancement that SpeQuloS aims to provide to BE-DCI users is execution stability. The execution stability is the ability to observe similar BoT completion times on the same execution environment (i.e., the BE-DCI considered, BoT workload, and DG middleware used). Providing a stable execution allows users to deduce from previous executions the QoS level they can expect from a BE-DCI. Figures 9(a) and 9(b) show the repartition functions of normalized BoT completion times around the average. Each execution completion time is divided by the average completion time measured under the same execution environment in terms of BE-DCI availability traces, DG middleware used, and BoT category. Figures report on results obtained with every BE-DCI traces and BoT categories mixed.

For the XWHEP middleware, the execution stability is not much improved by SpeQuloS, as it was already good without it. However, the execution stability of BoTs using BOINC middleware is significantly improved by SpeQuloS. Without SpeQuloS, Fig. 9(a) shows that a high number of executions have a normalized completion time lower than 1. This means that the average completion time is increased by a few, lengthy executions. As SpeQuloS is able to avoid such problematic cases, the average completion time becomes much more representative. This leads to very satisfactory execution stability, actually better than for XWHEP.

4.3.3 *Completion time prediction*

Table 7 shows the percentage of successful SpeQuloS predictions, described in Sect. 3.4, made when the BoT completion is 50 %. A successful prediction is reported when the actual completion time matches the SpeQuloS predicted time associated with an uncertainty of ±20 % (meaning that the actual completion time is comprised between 80 % and 120 % of the predicted time). For each BoT execution profiled, the α factor is computed using all available BoT executions with same BE-DCI trace, middleware, and BoT category. In other words, the "learning phase" (during which α is adjusted), is discarded and we assume perfect knowledge of the history of previous BoT executions.

Results show that the success rate of SpeQuloS prediction is high, except for some execution environments for which prediction is an issue. Still, the overall success rate is higher than 90 %, meaning than the predicted completion time given by SpeQuloS is correct within ±20 % in 9 cases out of 10, which is remarkable given the unpredictable nature of BE-DCIs. Results also show that predictions are slightly better with BOINC middleware than with XtremWeb-HEP, which can be explained by the more stable execution of this middleware, as reported in previous section. Another observation is that the RANDOM BoTs gives

Table 7 Percentage of success for SpeQuloS completion time prediction, according to BoT execution environment. A successful prediction is reported when the actual BoT completion time is comprised between ±20 % of the predicted completion time

BE-DCI	BoT category & middleware						
	SMALL		BIG		RANDOM		Mixed
	BOINC	XWHEP	BOINC	XWHEP	BOINC	XWHEP	
seti	100	100	100	82.8	100	87.0	94.1
nd	100	100	100	100	100	96.0	99.4
g5klyo	88.0	89.3	96.0	87.5	75	75	85.6
g5kgre	96.3	88.5	100	92.9	83.3	34.8	83.3
spot10	100	100	100	100	100	100	100
spot100	100	100	100	100	76	3.6	78.3
Mixed	97.6	96.1	99.2	93.5	89.6	65.3	90.2

inferior prediction quality. Indeed, as this BoT is highly heterogeneous, predicting completion time is harder as task execution times vary greatly amongst BoT executions.

Results of this section have shown that SpeQuloS is able to effectively enhance the QoS of BoTs executed on BE-DCIs. Indeed, using SpeQuloS, BoT completion time is accelerated by a factor of as much as 5, while assigning to Cloud resources less than 2.5 % of the total workload. Additionally, SpeQuloS increases the execution stability, meaning that BoTs executed in similar environments will present similar performance. Finally, SpeQuloS can accurately predict the BoT completion time and provide this information to BE-DCI users.

5 SpeQuloS use cases

5.1 SpeQuloS deployment in EDGI

In this section, we present the deployment of SpeQuloS as a part of the European Desktop Grid Infrastructure [14] (EDGI). EDGI connects several private and public Desktop Grids (IberCivis, University of Westminster, SZTAKI, CNRS/University of Paris XI LAL and LRI DGs) to several Grids (European Grid Infrastructure (EGI), Unicore, ARC) and private Clouds (StratusLab and local OpenStack, OpenNebula).

The main objective of EDGI is to transparently provide the vast amount of computing power of DGs to EGI users. Ultimately, these users would submit their applications to regular Computing Elements and thanks to EDGI, these tasks can be executed on DGs without any difference noticed by the user. SpeQuloS is one element amongst a full software stack, featuring a bridge from Grids to Desktop Grids, a data distribution network, monitoring framework, e-Science portal and more.

We present the current preliminary deployment of SpeQuloS, on part of the EDGI production infrastructure, which is illustrated in Fig. 10. The current deployment includes a production infrastructure, composed of two DGs, XW@LRI and XW@LAL, both ran by XWHEP and managed by the University of Paris-XI. For testing purposes, XW@LRI is connected to Grid5000 and gathers resources in best effort mode from 6 of its clusters with a bound on 200 nodes at a time. SpeQuloS uses Amazon EC2 as a supporting Cloud for XW@LRI. The XW@LAL server is connected to the local Desktop Grid of the laboratory. XW@LAL can also harvest computing resources from the EGI Grids through the EDGI 3G Bridge [37]. If a BoT is submitted to XW@LAL with the user' proxy certificate allowing an access to the Grid resources, then XW@LAL will detect it and launch corresponding PilotJob under user identity so that the whole Grid security and accounting chain is not broken. A user submitting BoT to XW@LAL without the proper Grid credential will have his BoT computed by the local Desktops. A local OpenNebula part of the StratusLab infrastructure is used as a supporting Cloud for the LAL Desktop Grid.An interesting side-effect of this setup is that BoTs submitted through XtremWeb-HEP to EGI can eventually benefit from the QoS support provided by SpeQuloS using resources from StratusLab. In the context of the EDGI project, another SpeQuloS deployment provides QoS support to other EDGI DGs, such as SZTAKI's one, through a fully-dedicated OpenNebula Cloud service.

Several EDGI applications are installed and used regularly, such as DART (a Framework for Distributed Audio Analysis and Music Information Retrieval by Cardiff University), BNB-Grid (which is aimed at solving hard combinatorial, discrete and global optimization problems) and ISDEP (which is a fusion plasma application which simulates the Tokamak of ITER). Table 8 summarizes the usage of the infrastructure during the first half of 2011 where SpeQuloS has been gradually deployed.

5.1.1 Experimentation in the EDGI testing infrastructure

To test SpeQuloS implementation and evaluate its performance under realistic conditions, we performed experimentations using Grid5000 [8].

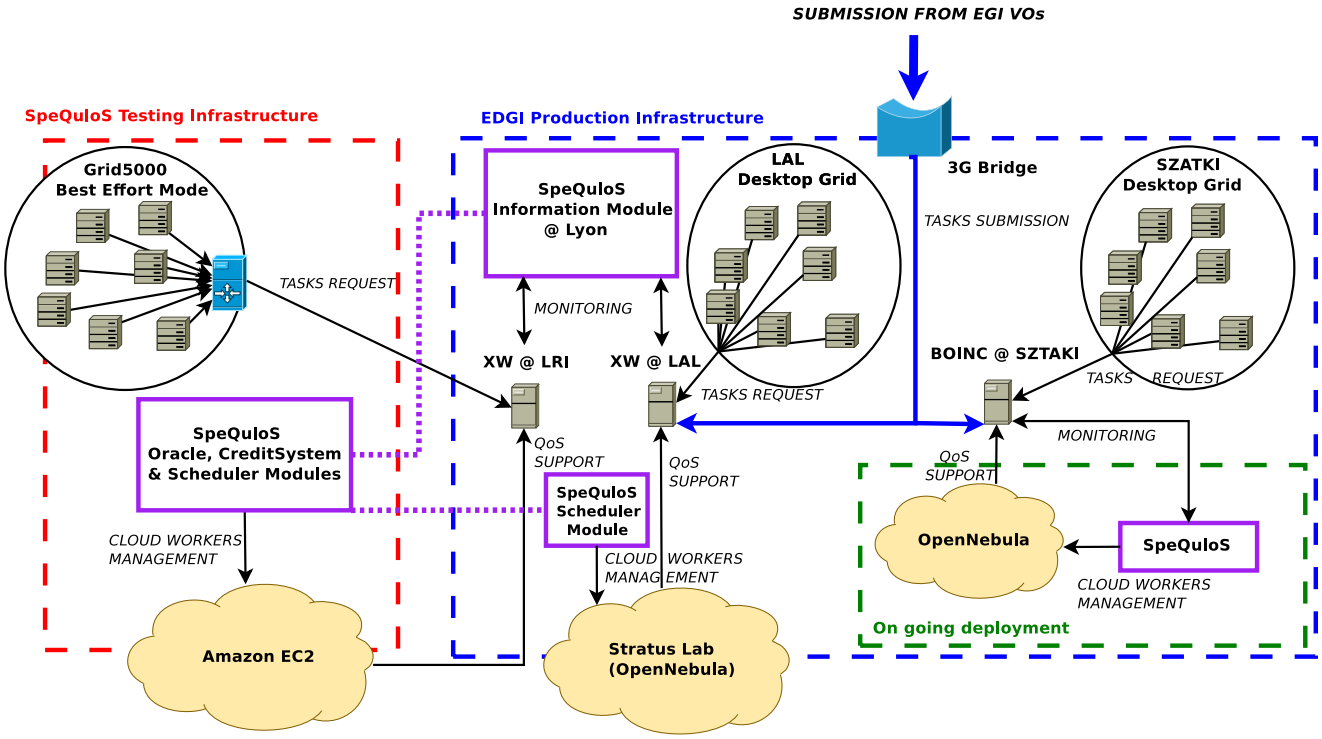


Fig. 10 SpeQuloS' current deployment as a part of the EDGI infrastructure. SpeQuloS' modules are split and duplicated across the deployment

Table 8 The University Paris-XI part of the European Desktop Grid Infrastructure. The table reports on the number of tasks executed on XW@LAL and XW@LRI Desktop Grids, as well as the number of EGI tasks executed on those DGs and the number of tasks assigned by SpeQuloS to StratusLab and Amazon EC2 Cloud services

	XW@LAL	XW@LRI	EGI	StratusLab	EC2
#tasks	557002	129630	10371	3974	119

In usual Desktop Grids, from few hundreds to several thousands worker nodes are participating to computational effort during several days. To reproduce this situation, we used Grid5000 resources as Desktop Grids workers. Our experimentation is supported by the XtremG5K project, which main goal is to use Grid5000 to provide computational resources to an XWHEP server. The XtremG5K project is composed of several components:

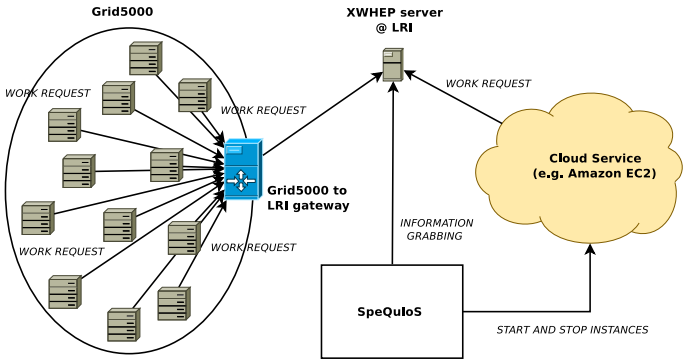
– The XWHEP XW@LRI Desktop Grid server. It can accept jobs submission from EGEE users through the 3Gbridge middleware. Unfortunately, at the time of the experiment, the 3Gbridge was not functioning and we had to use the XWHEP server as an entry point. However, this does not impact the SpeQuloS scenarios presented here. The XWHEP server is in charge of jobs distribution among XWHEP workers, and results collection.
– A gateway, administrated in Grid5000, which enables communications between Grid5000 network and the XW@LRI server.
– A set of XWHEP workers, executed on Grid5000 nodes. In this experimentation, a Grid5000 job is considered as a pilot job which runs one or several XWHEP workers. To approximate voluntary-based and best effort participation to computation as observed with DG worker nodes, we used the "best effort" queue of Grid5000.

Figure 11(a) shows the experimentation architecture.

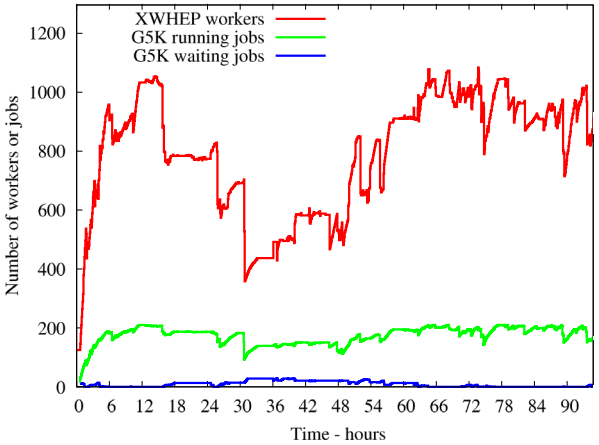
5.2 Grid5000 as a Best-Effort DCI

A solution for Grid5000 users who want their BoT to complete within a certain time is to reserve a block of nodes for a given time and to launch Pilot Jobs to execute the BoT. Although this approach is suitable for the users, these rigid reservations lead to an inefficient management of the platform. By using Grid50000 as a Best-Effort DCI combined with SpeQuloS allows giving a more satisfactory experience to the user while keeping flexible resource management.

The experimentation has been deployed on seven Grid5000 clusters (from Lyon, Grenoble, Bordeaux, Lille, Nancy, Sophia and Toulouse sites). On each of them, we ran Algorithm 3 for Pilot Jobs submission. Each Pilot Job, denoted as `start_XWHEP_workers` in the algorithm, is submitted to one Grid5000 node and starts one XWHEP worker per available CPU.



(a) The XtremG5K project and SpeQuloS architecture



(b) Number of XWHEP workers and Grid5000 pilot jobs running and waiting during the experimentation

Fig. 11 The XtremG5K project

Algorithm 3 Algorithm for Pilot Jobs submission executed on each Grid5000 site

```
max_#pjobs ← 30
max_#pjobs_waiting ← 7
while true do
  if current_#pjobs < max_#pjobs then
    if current_#pjobs_waiting < max_#pjobs_waiting
    then
      "Submit start_XWHEP_workers to one Grid5000
      node in best effort queue"
    else
      "Too many pilot jobs waiting"
    end if
  else
    "Maximum number of pilot jobs reached"
  end if
  sleep(15 minutes)
end while
```

We ran this experimentation during 4 days. Figure 11(b) shows the number of pilot jobs, running or waiting for submission in Grid5000, and the corresponding number of XWHEP workers, according to time of experimentation. We can see that the total number of Grid5000 pilot jobs was comprised between 100 and 200, which leads to 400–1000 workers to participate to the DG. The number of Grid5000 pilot jobs waiting for submission on all sites simultaneously never exceeds 30.

We ran several BoT execution scenarios with or without using SpeQuloS. In each scenario, an XWHEP user submits a BoT containing from 1000 to 10000 jobs. Each individual job typically needs few minutes to be computed by a worker. Jobs are distributed by the XW@LRI server to nodes hosted in Grid5000 and to Cloud workers from Amazon EC2 (when SpeQuloS was used).

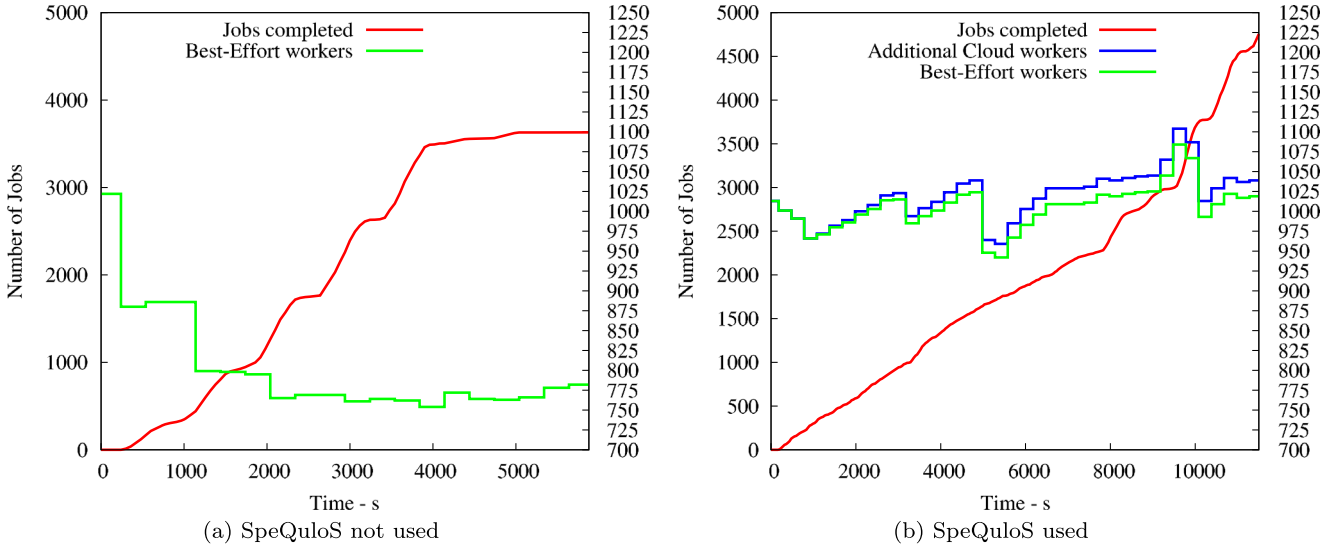
Figure 12(a) shows result from a scenario where SpeQuloS is not used. A BoT of 3630 jobs is submitted at the beginning of the scenario. The figure shows the number of jobs completed as well as the number of DG workers, according to the scenario time. The number of jobs completed denotes the number of jobs for which execution by DG node has been completed, and results sent back to the LRI server.

We can see that the BoT completion increases quickly during the first 4000th seconds of the scenario. Then, the completion grows slowly while all jobs have been distributed to workers. This figure illustrates the "tail effect" of BoT completion in DG, discussed in Sect. 2.2.

By provisioning stable resources from Cloud, one goal of SpeQuloS is to address this problem. Figure 12(b) shows a similar scenario, at the difference that SpeQuloS is used. In this scenario, a BoT of 4750 jobs was submitted. In addition to previous scenario results, the figure shows the Cloud workers started by SpeQuloS to participate to the DG. The amount of cloud resources available to SpeQuloS was limited and equivalent to 60 CPU.hours of the "small" instance of Amazon EC2.

No tail effect can be observed in this scenario. The BoT completion increases regularly during the first 8000th seconds of the scenario and grows even faster afterwards. We can see that SpeQuloS, according to the policy used in its Scheduler module, starts Cloud workers which participate to the computation. The first worker is started at the 15th minute of the scenario. SpeQuloS then adds Cloud workers gradually until the maximum of 20 simultaneous workers is reached.

Additional experimentations are needed to draw any general conclusion on the accordance of the SpeQuloS implementation with the simulation results presented earlier.



(a) SpeQuloS not used (b) SpeQuloS used

Fig. 12 Completion of jobs with number of desktop grid workers, according to the scenario time

However, the experimentations presented in this section demonstrate the ability to use Grid5000 to run large scale experiments of hybrid infrastructure, mixing DG and Cloud services in order to experiment SpeQuloS in realistic conditions.

5.3 Grid5000 as a Cloud

To evaluate the benefits of Cloud resources provisioning in a large scale scenario, a large amount of resources from Cloud must be available to experiments. However, those resources are not always available to researchers because of the cost of public Cloud services such as Amazon EC2 and the complexity to build and maintain private Cloud services using technologies such as Eucalyptus or OpenNebula. To address this issue, we decided to use Grid5000 as a new type of Cloud available to SpeQuloS. Indeed, Grid5000 can provide on-demand computing resources and includes most of the features provided by commercial Cloud services.

As SpeQuloS uses the libcloud API as a common interface to interact with various types of Cloud services, we decided to use the Grid5000 API[2] to implement a new libcloud driver to access to Grid5000 resources. This implementation allows interaction with Grid5000 as a typical IaaS Cloud service, with the same API as any other Cloud service for which driver exists in libcloud. With the Grid5000 libcloud driver, a large amount of Cloud resources is made available for experiments. It is also possible to combine Grid5000 and other Cloud services in the same experiment.

To validate the Grid5000 libcloud driver, we ran an experimentation scenario with SpeQuloS provisioning Cloud resources from both Amazon EC2 and Grid5000. The rest of the experiment environment is similar to that presented in Sect. 5.2.

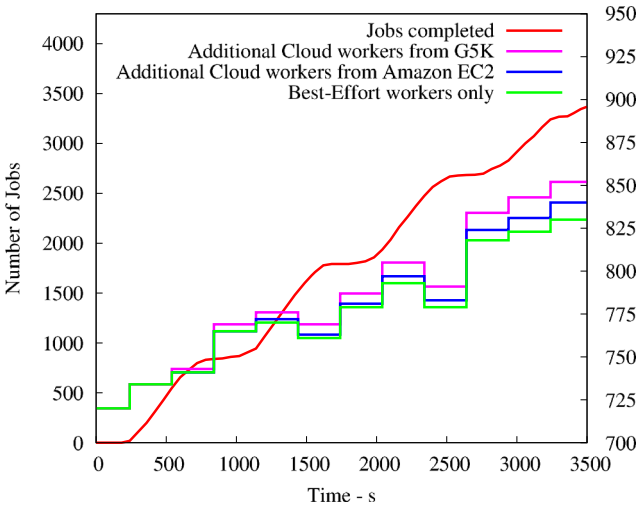


Fig. 13 Completion of jobs with number of desktop grid workers, according to the scenario time. Grid5000 resources are used as Cloud workers

Figure 13 shows results from a scenario where SpeQuloS uses, in addition to Amazon EC2 resources, some Grid5000 resources as Cloud workers. In this scenario, up to 15 Grid5000 Cloud workers are started by SpeQuloS to participate to the DG, each Cloud worker running on a Grid5000 node hosted in the Rennes site.

5.4 Using spot instances and SpeQuloS to decrease BoT execution costs on Amazon EC2

This scenario features a low cost usage of Amazon EC2. User wants to execute a BoT on Cloud resources from Ama-

[2] https://api.grid5000.fr/.

zon EC2. Instead of regular instances, spot instances are provisioned to decrease the total cost. As spot instances are not as reliable as regular instances, SpeQuloS is used to bring similar level of QoS for the BoT execution.

We deployed an experimentation implementing this scenario for a period of 7 days and bid to instantiate the maximum number of spot instances, according to the spot market price, for a renting cost of 10 dollars per hour. In the experimentation, we used the *c1.large* instance. BOINC middleware executes the BoT which is composed of 10000 tasks of the DSP application. The BOINC server is deployed on a regular instance and each spot instance ran a BOINC client that connects to this server. SpeQuloS modules are deployed on the same host than the BOINC server. We configured SpeQuloS to start Amazon EC2 regular instances as Cloud worker to support the BoT execution.

The BoT execution took 6.65 hours to complete. The total cost of the spot instances was 70 dollars and the cost of the instance which ran the server was 2.4 dollars. As SpeQuloS started 20 instances during 9 minutes at the end of the BoT execution and one hour of cloud usage is billed for each of these instances, the total cost of the SpeQuloS cloud worker is 6.8 dollars. Hence, the total cost of the execution is 79.2 dollars. Executing the BoT on the same amount of resource, but using regular instances instead of spot instances would cost 206.7 dollars. Using spot instances and SpeQuloS allows cutting more than twice the cost of BoT execution on Amazon EC2, while preserving the same level of QoS, thanks to SpeQuloS features.

6 Related work

Many scenarios motivate the assemblage of Grids or Clouds with Best Effort infrastructures, and in particular Desktop Grids. GridBot [36] puts together Superlink@Technion, Condor pools and Grid resources to execute both throughput and fast-turnaround oriented BoTs. The European FP7 projects EDGeS [37] and EDGI [14] have developed bridge technologies to make Desktop Grid infrastructure transparently available to any EGI Grid users as a regular Computing Element. Similarly, the Latin America EELA-2 Grid has been bridged with the OurGrid infrastructures [9].

In [34], authors investigate the cost and performance of running a Grid workload on Amazon EC2 Cloud. Similarly, in [25], the authors introduce a cost-benefit analysis to compare Desktop Grids and Amazon EC2. ElasticSite [29] offloads a part of the Grid workload to the Cloud when there is peak user demand. In [1], authors propose a Pareto efficient strategy to offload Grid BoTs with deadlines on the Cloud.

Providing QoS features in Grids is hard and not solved yet satisfactorily [13, 21, 39]. It is even more difficult in an environment where there are no guaranteed resources [7]. Unlike aforementioned work, we do not modify the resource manager scheduling policies to incorporate QoS features. Instead, we use an extrinsic approach by providing additional resources. However, the two approaches could coexist by classifying the DG workers according to their historical behavior and allocating applications with QoS needs to the more trustable and faster workers. In [38], a framework is presented to extend Grid resources using Cloud computing. Similarly, Aneka [10] supports the integration between Desktop Grids and Clouds. These works would be the closest to ours although we went further in term of implementation and evaluation.

There exists a large literature about predicting tasks completion time. For instance QBETS [32] uses time series to model and forecast task queues. Closer to our context, [15] proposes a framework to model and predicts the various steps (submission, validation, waiting in the scheduler queue) that a work unit spend in a volunteer computing project. Our work differs by the fact that we address heterogeneous environments. As a result, we adopted a unique representation based on BoT progression to hide idiosyncrasies of BE-DCIs. Thus, the Oracle never accesses directly the BoT Queue, but rather a history of past BoTs and on-line monitoring information.

Mitigation of the tail in Desktop Grid computing has been addressed in the past [23]. The difference between that prior work and ours is that we provide prediction and stability estimates for QoS, we devise new algorithms for using dedicated cloud resources, and we evaluate these algorithms more completely in a wide range of scenarios (in terms of different BoT classes, desktop grid middleware, and platforms with different degrees of volatility and heterogeneity).

In [40], authors propose the LATE (Longest Approximate Time to End) scheduling to alleviate outliers in MapReduce computation. The LATE scheduler monitors tasks execution and speculatively executes those of the tasks which are anticipated to have the latest finished time on the fastest hosts. Recently, the Mantri system [3] have been proposed, where the authors identifies several causes of dramatic slowdown of computation, including workload imbalance due to data skew, network contention due to disadvantageous communication patterns and overloaded machine. Because these MapReduce systems run within a cluster, they assume a finer grain of information: individual task monitoring versus global BoT progress rate monitoring in the case of SpeQuloS. SpeQuloS deals with considerably large infrastructures, potentially hundreds of thousands hosts with very different characteristics in the case of Desktop Grids. As infrastructures are treated as black box, SpeQuloS cannot implement MapReduce speculative execution heuristics which relies on a per-hosts information or network topologies information in the case of Mantri.

Providing cost-effective usage of Cloud resources is a topic of growing interest. Authors of [27] propose a mechanism to minimize the cost of scheduling an entire workflow on Cloud resources, while trying to satisfy a user-supplied deadline. Conversely, [33] presents a scheduler that minimizes completion time of BoT executed on multiple Clouds under a constrained budget. In our work, most of workload is processed by BE-DCIs and we only use Cloud resources to process its most critical part. However, these works could be considered to optimize Cloud resources usage by SpeQuloS.

In the paper [12], we have presented SpeQuloS design and performances. We augment this publication by showing several new use-cases and the experimental performance evaluation associated. In addition, we give the user more details about the simulation evaluation and the framework itself.

7 Conclusion and future works

We have introduced SpeQuloS, a service to enhance QoS for BoT applications when executed in hybrid DCIs. By taking advantage of their elasticity properties, SpeQuloS allows Best Effort Distributed Computing Infrastructures (BE-DCIs) such as Desktop Grids, Best Effort Grids or Cloud Spot instances to become more widely accessible.

SpeQuloS monitors execution of BoTs and dynamically provisions reliable and efficient resources from Cloud to offload critical part of BoT execution from BE-DCIs. Several Cloud provisioning strategies were investigated and evaluated using trace-driven simulations. Although providing QoS to grid computing is considered as a difficult issue, our approach is able to substantially improve QoS according to several metrics, such as completion time, execution stability and prediction, and by providing feedback to users on the QoS benefits they can expect.

The context of hybrid infrastructures had consequences on SpeQuloS development and deployment. Our framework is composed of four different distributed modules dedicated to specific tasks: information, accounting, prediction and scheduling. We have presented three use cases to demonstrate versatility and effectiveness under various hybrid deployments. In particular, it is a key component of the European Desktop Grid Infrastructure, where the service will allow users to have a similar experience when using Desktop Grids than computing on regular DCIs.

Our future work will focus on improving the performance of tail detection and mitigation. In particular, we would like to anticipate when a BoT is likely to produce a tail by correlating the execution with the state of the infrastructure: resource heterogeneity, variation in the number of computing resources and rare events such as massive failures or network partitioning.

Acknowledgements Authors would like to thank Peter Kacsuk, Jozsef Kovacs, Michela Taufer, Trilce Estrada and Kate Keahey for their insightful comments and suggestions throughout our research and development of SpeQuloS.

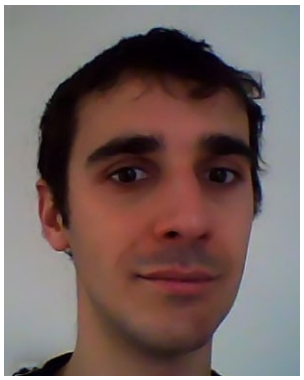
Some of the experiments presented in this paper were carried out using the Grid5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

This work was funded by the EDGI project, supported by the European Commission FP7 Capacities Programme under grant agreement RI-261556.

References

1. Agmon Ben-Yehuda, O., Schuster, A., Sharov, A., Silberstein, M., Iosup, A.: ExPERT: Pareto-efficient task replication on grids and clouds. Technical Report CS-2011-03, Technion (2011)
2. Amazon Web Services: An introduction to spot instances. Technical Report, Amazon Elastic Compute Cloud (2009)
3. Ananthanarayanan, G., Kandula, S., Greenberg, A., Stoica, I., Lu, Y., Saha, B., Harris, E.: Reining in the outliers in map-reduce clusters using Mantri. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10 (2010)
4. Anderson, D.: BOINC: a system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International GRID Workshop, Pittsburgh, USA (2004)
5. Andrade, N., Brasileiro, F., Cirne, W., Mowbray, M.: Automatic grid assembly by promoting collaboration in peer-to-peer grids. J. Parallel Distrib. Comput. **67**(8), 957–966 (2007)
6. Andrade, N., Cirne, W., Brasileiro, F., Roisenberg, P.: OurGrid: an approach to easily assemble grids with equitable resource sharing. In: Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing (2003)
7. Anglano, C., Brevik, J., Canonico, M., Nurmi, D., Wolski, R.: Fault-aware scheduling for bag-of-tasks applications on desktop grids. In: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, GRID '06 (2006)
8. Bolze, R., et al.: Grid5000: a large scale highly reconfigurable experimental grid testbed. Int. J. High Perform. Comput. Appl. **20**(4), 481–494 (2006)
9. Brasileiro, F., Duarte, A., Carvalho, D., Barber, R., Scardaci, D.: An approach for the co-existence of service and opportunistic grids: the EELA-2 case. In: Latin-American Grid Workshop (2008)
10. Calheiros, R.N., Vecchiola, C., Karunamoorthy, D., Buyya, R.: The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid clouds. Future Gener. Comput. Syst. **28**(6), 861–870 (2011)
11. Capit, N., Da Costa, G., Georgiou, Y., Huard, G., Martin, C., Mounie, G., Neyron, P., Richard, O.: A batch scheduler with high level components. In: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), Washington, DC, USA (2005)
12. Delamare, S., Fedak, G., Kondo, D., Lodygensky, O.: SpeQuloS: a QoS service for BoT applications using best effort distributed computing infrastructures. In: Proceedings of the 21st ACM International Symposium on High Performance Distributed Computing (HPDC'12), Delft, The Netherlands, pp. 173–186 (2012)
13. Dong, F., Akl, S.G.: Scheduling algorithms for grid computing: State of the art and open problems. Technical Report, Queen's University Kingston (2006)
14. European desktop grid infrastructure (2010). http://edgi-project.eu/

15. Estrada, T., Reed, K., Taufer, M.: Modeling job lifespan delays in volunteer computing projects. In: 9th IEEE International Symposium on Cluster Computing and Grid (CCGrid) (2009)
16. Fedak, G., Germain, C., Neri, V., Cappello, F.: XtremWeb: a generic global computing platform. In: CCGRID'2001 Special Session Global Computing on Personal Devices (2001)
17. Fishelson, M., Geiger, D.: Exact genetic linkage computations for general pedigrees. Bioinformatics **18**(Suppl 1), S189–S198 (2002)
18. Heien, E., Kondo, D., David, A.: Correlated resource models of Internet end hosts. In: 31st International Conference on Distributed Computing Systems (ICDCS), Minneapolis, Minnesota, USA (2011)
19. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H.: The grid workloads archive. Future Gener. Comput. Syst. **24**(7), 672–686 (2008)
20. Iosup, A., Sonmez, O., Anoep, S., Epema, D.: The performance of bags-of-tasks in large-scale distributed systems. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing, HPDC '08 (2008)
21. Islam, M., Balaji, P., Sadayappan, P., Panda, D.: QoPS: a QoS based scheme for parallel job scheduling. In: Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science. Springer, Berlin (2003)
22. Javadi, B., Kondo, D., Vincent, J., Anderson, D.: Mining for statistical availability models in large-scale distributed systems: an empirical study of SETI@home. In: 17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) (2009)
23. Kondo, D., Chien, A., Casanova, H.: Resource management for rapid application turnaround on enterprise desktop grids. In: ACM Conference on High Performance Computing and Networking, SC 2004, USA (2004)
24. Kondo, D., Javadi, B., Iosup, A., Epema, D.: The Failure Trace Archive: enabling comparative analysis of failures in diverse distributed systems. In: 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (2010)
25. Kondo, D., Javadi, B., Malecot, P., Cappello, F., Anderson, D.: Cost-benefit analysis of cloud computing versus desktop grids. In: 18th International Heterogeneity in Computing Workshop (2009)
26. Litzkow, M., Livny, M., Mutka, M.: Condor—a hunter of idle workstations. In: Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS) (1988)
27. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11. ACM, New York (2011)
28. Marosi, A.C., Kacsuk, P.: Workers in the clouds. In: Euromicro Conference on Parallel, Distributed, and Network-Based Processing (2011)
29. Marshall, P., Keahey, K., Freeman, T.: Elastic site: using clouds to elastically extend site resources. In: Proceedings of CCGrid'2010, Melbourne, Australia (2010)
30. Marshall, P., Keahey, K., Freeman, T.: Improving utilization of infrastructure clouds. In: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011) (2011)
31. Minh, T.N., Wolters, L.: Towards a profound analysis of bags-of-tasks in parallel systems and their performance impact. In: High-Performance Parallel and Distributed Computing (2011)
32. Nurmi, D.C., Brevik, J., Wolski, R.: QBETS: queue bounds estimation from time series. In: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '07 (2007)
33. Oprescu, A.M., Kielmann, T.: Bag-of-tasks scheduling under budget constraints. In: CloudCom (2010)
34. Palankar, M.R., Iamnitchi, A., Ripeanu, M., Garfinkel, S.: Amazon S3 for science grids: a viable solution? In: Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing, DADC '08 (2008)
35. Rood, B., Lewis, M.J.: Multi-state grid resource availability characterization. In: 8th Grid Computing Conference (2007)
36. Silberstein, M., Sharov, A., Geiger, D., Schuster, A.: GridBot: execution of bags of tasks in multiple grids. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09 (2009)
37. Urbah, E., Kacsuk, P., Farkas, Z., Fedak, G., Kecskemeti, G., Lodygensky, O., Marosi, A., Balaton, Z., Caillat, G., Gombas, G., Kornafeld, A., Kovacs, J., He, H., Lovas, R.: EDGeS: bridging EGEE to BOINC and XtremWeb. J. Grid Comput. **7**, 335–354 (2009)
38. Vázquez, C., Huedo, E., Montero, R.S., Llorente, I.M.: On the use of clouds for grid resource provisioning. Future Gener. Comput. Syst. **27**(5), 600–605 (2011)
39. Weng, C., Lu, X.: Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid. Future Gener. Comput. Syst. **21**(2), 271–280 (2005)
40. Zaharia, M., Konwinski, A., Joseph, A., Katz, R., Stoica, I.: Improving MapReduce performance in heterogeneous environments. In: OSDI'08 (2008)



Simon Delamare is a research engineer from CNRS, working inside the LIP laboratory at ENS Lyon. He received his Master at Université Paris 6 and his Ph.D. at Telecom ParisTech in 2010. His research interests include quality of service and reliability of distributed computing infrastructures.

Gilles Fedak has been a permanent INRIA research scientist since 2004 and is currently working in the AVALON team. After graduated from University Paris Sud in 2003, he has followed a postdoctoral fellowship at University California San Diego in 2003–2004. His research topics encompass parallel and distributed computing with a focus on resource management, large scale data processing, security and reliability.

Derrick Kondo received his Bachelor's at Stanford University in 1999, and his Master's and Ph.D. at the University of California at San Diego in 2005, all in computer science. His general interests are in the areas of reliability, fault-tolerance, statistical analysis, job and resource management.

Oleg Lodygensky is a research engineer at CNRS, working at LAL (a high energy physics laboratory) since 1996. He had his Master at Université Paris 6 and defended his PhD at Université Paris 11. He works on research in distributed computing and is the project manager of XtremWeb-HEP, a large scale computing platform, involved in major european projects.