SpeQuloS: A QoS Service for BoT Applications Using Best Effort Distributed Computing Infrastructures.

Simon Delamare, Gilles Fedak, Derrick Kondo, Oleg Lodygensky

SpeQuloS: A QoS Service for BoT Applications Using Best Effort Distributed Computing Infrastructures

Simon Delamare , Gilles Fedak , Derrick Kondo , Oleg Lodygensky

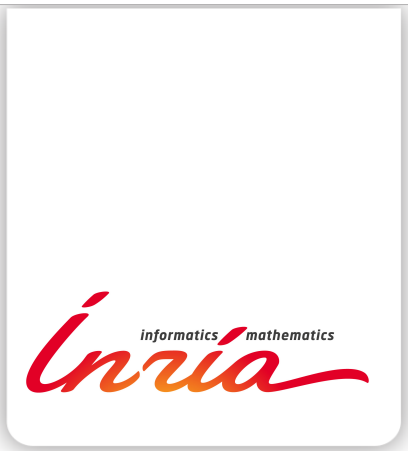
RESEARCH REPORT

N° 7890

February 2012

Project-Team GRAAL

ISSN 0249-6399 ISRN INRIA/RR--7890--FR+ENG



SpeQuloS: A QoS Service for BoT Applications Using Best Effort Distributed Computing Infrastructures

Simon Delamare [*], Gilles Fedak [†], Derrick Kondo [‡], Oleg Lodygensky [§]

Project-Team GRAAL

Research Report n° 7890 — February 2012 — 22 pages

Abstract: Exploitation of Best Effort Distributed Computing Infrastructures (BE-DCIs) allow operators to maximize the utilization of the infrastructures, and users to access the unused resources at relatively low cost. Because providers do not guarantee that the computing resources remain available to the user during the entire execution of their applications, they offer a diminished Quality of Service (QoS) compared to traditional infrastructures. Profiling the execution of Bag-of-Tasks (BoT) applications on several kinds of BE-DCIs demonstrates that their task completion rate drops near the end of the execution.
In this report, we present the SpeQuloS service which enhances the QoS of BoT applications executed on BE-DCIs by reducing the execution time, improving its stability, and reporting to users a predicted completion time. SpeQuloS monitors the execution of the BoT on the BE-DCIs, and dynamically supplies fast and reliable Cloud resources when the critical part of the BoT is executed. We present the design and development of the framework and several strategies to decide when and how Cloud resources should be provisioned. Performance evaluation using simulations shows that SpeQuloS fulfill its objectives. It speeds-up the execution of BoTs, in the best cases by a factor greater than 2, while offloading less than 2.5% of the workload to the Cloud. We report on preliminary results after a complex deployment as part of the European Desktop Grid Infrastructure.

Key-words: Distributed Computing, Quality of Service, Bag of Tasks

[*] INRIA/University of Lyon - Simon.Delamare@inria.fr
[†] INRIA/University of Lyon - Gilles.Fedak@inria.fr
[‡] INRIA/University of Joseph Fourier - Derrick.Kondo@inria.fr
[§] IN2P3/University of Paris XI - Oleg.Lodygensky@lal.in2p3.fr

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

SpeQuloS: Une architecture pour la QoS des applications BoT dans les infrastructures de calcul distribué «Best-Effort»

Résumé : L'utilisation des infrastructures de calcul distribué «Best-effort» (BE-DCI) permet aux opérateurs de maximiser l'utilisation de leurs infrastructures, et aux utilisateurs d'accéder aux ressources inutilisées à moindre coût. La disponibilité de ces ressources ne pouvant être garantie tout au long de l'exécution d'une application, elles offrent une qualité de service (QoS) moindre que les infrastructures classiques. En étudiant l'exécution d'applications «sac de taches» (BoT) sur différentes BE-DCIs, nous observons que le taux de complétion des BoTs chute fortement à la fin de leur exection.

Dans ce document, nous présentons SpeQuloS, une architecture qui améliore la QoS des applications BoT exécutées sur les BE-DCIs en diminuant leurs durées de complétion, en augmentant leurs prévisibilités et en informant les utilisateurs du temps restant prédit. SpeQuloS surveille l'exécution d'un BoT sur une BE-DCI et déploie dynamiquement des ressources stables et fiables issues du Cloud lorsque la partie critique du BoT est exécutée. Nous présentons l'architecture et l'implémentation de notre prototype ainsi que plusieurs stratégies pour décider quand et combien de ressources Cloud devraient être utilisées. Les évaluations de performances réalisées à l'aide de simulations montrent que SpeQuloS remplit ses objectifs. Il accélère jusqu'à deux fois le temps de complétion des BoTs en ne faisant exécuter que 2,5% de la charge de travail sur le Cloud. Nous présentons également des résultats préliminaires issus d'un déploiement complexe au sein de l'European Desktop Grid Infrastructure.

Mots-clés : Calcul distribué, Qualité de Service, Sac de Tâches

1 Introduction

There is a growing demand for computing power from scientific communities to run large applications and process huge volumes of scientific data. Meanwhile, Distributed Computing Infrastructures (DCIs) for scientific computing continue to diversify. Users can not only select their preferred architectures amongst Supercomputers, Clusters, Grids, Clouds, Desktop Grids and more, based on parameters such as performance, reliability, cost or quality of service, but can also combine transparently several of these infrastructures together. The quest for more computing power also depends on the emergence of infrastructures that would both offer lower operating costs and larger computing power.

Amongst the existing DCIs, *Best Effort* DCIs are able to meet these two criteria. We call Best Effort DCIs (BE-DCIs) an infrastructure or a particular usage 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[?], OurGrid[?], XtremWeb[?]) and Volunteer Computing Systems (BOINC[?]), which rely on idle desktop PCs are typical examples of Best Effort DCIs. But, one can think of other examples: Grid resource managers such as OAR[?] manage a best effort queue to harvest idle nodes of the cluster. 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. In Cloud computing, Amazon has recently introduced EC2 Spot instances[?] where users can bid for unused Amazon EC2 instances. If the market Spot price goes under the user's bid, a user gains access to available instances. Conversely when the Spot price exceeds his bid, the instance is terminated without notice. Similar features exist in other Cloud services[?].

Although 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 [?] or SuperLink [?] propose technologies to make BE-DCIs, in particular Desktop Grids, available to Grid users as regular computing element such as clusters.

However, because BE-DCIs trade reliability against lower prices, 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 the Grid [?], 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. 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.

To achieve this objective, SpeQuloS dynamically deploys fast and trustable workers from Clouds that are available to support the BE-DCIs. The issue of outliers tasks slowing down the executions has been addressed by the Mantri system for MapReduce applications [?]. 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 to deliver SpeQuloS as an on-line multi-BoT, multi-users service and able to serve several BE-DCI simultaneously. In this paper, we describe strategies based on BoT completion thresholds and task execution variance for deciding when to assign tasks to Cloud workers. We also investigate two approaches for Cloud resource provisioning and different methods for workload assignment on Cloud resources. We evaluate these strategies using trace-driven simulations based from actual Grid, Cloud, and Desktop Grid infrastructures. Our simulator models two middleware which represents two different approaches for handling hosts volatility. One middleware, called XtremWeb-HEP (XWHEP), uses host failure detection based on heartbeats. The second middleware, called BOINC, uses task deadlines and task replication.

Our simulation results show that SpeQuloS correctly addresses the tail effect: In half of the cases the tail has totally disappeared, in the other half it has been significantly reduced. As a

consequence, both for XtremWeb-HEP and BOINC the execution of BoT applications is greatly improved on every BE-DCIs investigated and for various kind of BoT workloads. Nevertheless, the strategies implemented are shown to make a minimal use of the Cloud resources: In the best cases where a speed-up greater than 2 is achieved, we observe that less 2.5% of the BoT has been offloaded to the Cloud. Finally, our evaluation shows that users experience can be greatly improved as success rate on the prediction of the BoT completion time is 90% on average.

We also describe the implementation and deployment of SpeQuloS in the scope of the European Desktop Grid Initiative FP7 project[?] (EDGI). Dealing with the deployment of a such a complex infrastructure had strong a consequence on the design of the service. Its architecture is modular and distributed through several independent components. It supports several Desktop Grid middleware (XtremWeb-HEP, BOINC) and Cloud technologies (Amazon EC2, OpenNebula, Nimbus, Rackspace) In this paper, we present the development of the prototype and some preliminary results after being deployed on part of the production European Desktop Grid Infrastructure (EDGI).

The rest of the paper is organized as follow. In Section 2, we introduce our analysis of running BoT applications on best effort infrastructures. The SpeQuloS framework is presented in Section 3. Section 4 presents performance evaluation. Section 5 reports on real-world deployment. Related works are presented in Section 6 and finally we conclude in Section 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[?], allow submission in a Best Effort queue. Tasks submitted to that queue have a lower priority and can be preempted by any other task. 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. An 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[?], 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 which 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 [?, ?, ?] follow a similar approach, and find it to be efficient and scalable.

We captured several BoT executions, using the experimental environment described in Section 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 Figure 1: The last fraction of the BoT takes a large part of the total execution time. We called this phenomenon the **tail effect**.

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 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 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 is to 400% with XWHEP and 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.

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

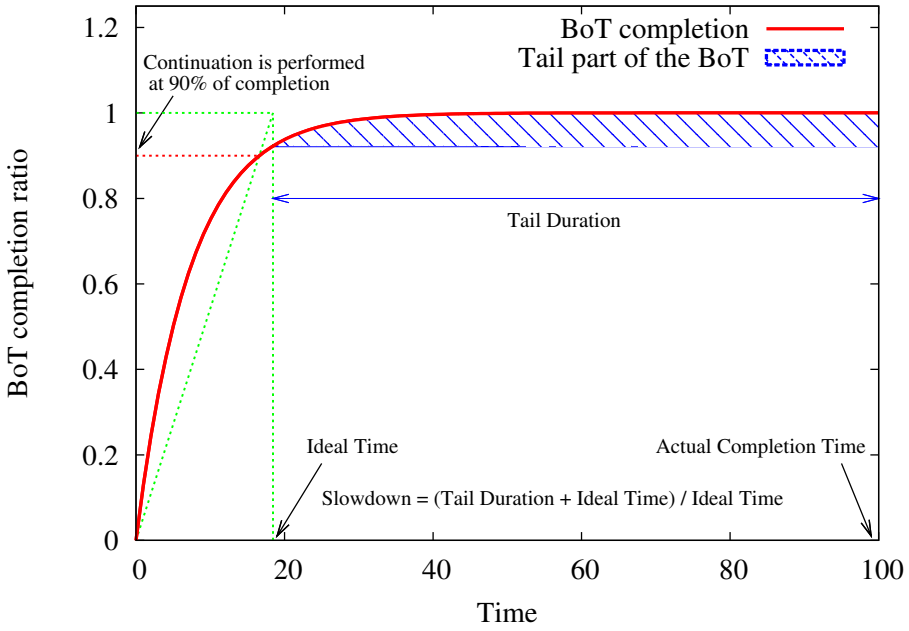


Figure 1: Example of BoT execution with noteworthy values

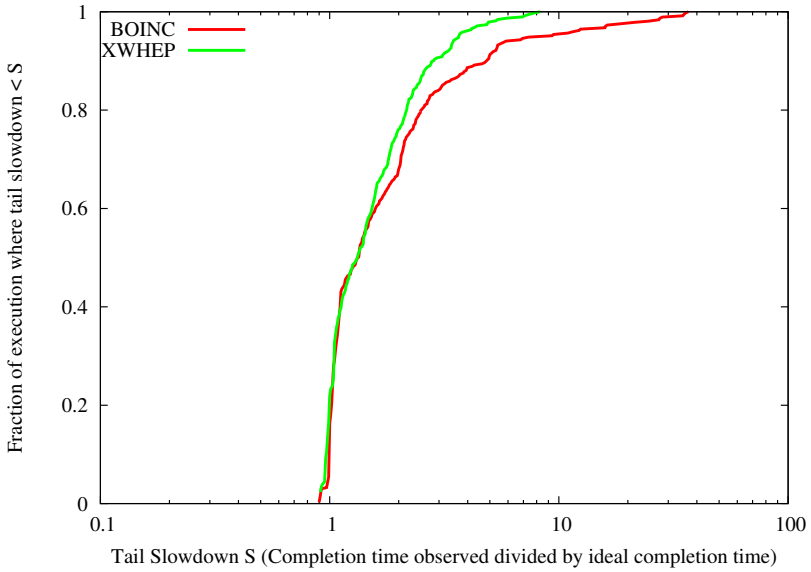


Figure 2: Profiling execution of BoTs in BE-DCIs: Tail Slowdown is the BoT completion time divided by the ideal completion time (i.e. determined by assuming a constant completion rate). The cumulative distribution function of observed slowdowns is represented.

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 total BoT execution time spent in the tail.

	Avg. % of BoT in tail		Avg. % of time in tail	
BE-DCI Trace	**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

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 an 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 XW-HEP 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 provide QoS to users of Best Effort DCIs managed by Desktop Grids (DG) middleware, by provisioning stable resources from Cloud services.

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 Figure 3. The Information module gathers and stores information from BE-DCIs (see Section 3.2). The Credit System module is in charge of the billing and accounting of Cloud-related operations (Section 3.3). The Oracle module helps SpeQuloS determine how to efficiently deploy the Cloud resources, and gives QoS information to users (Section 3.4 and 3.5). The Scheduler module manages the BoT and the Cloud workers during its execution (Section 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 requesting QoS to the Scheduler to support a BoT execution. The Scheduler returns an identifier (*BoTId*) that must be used for every trans-

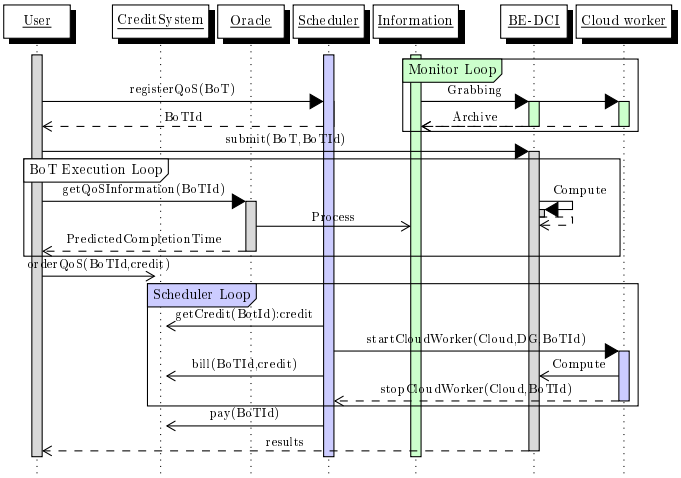


Figure 3: Sequence diagram of SpeQuloS interactions in a typical use-case scenario

action related to this BoT, and adds it to the list of BoTs it manages. Then, the user can submit the BoT tagged with the *BoTId* to the BE-DCI,

- 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 the BoTs it manages. If credits are provisioned for a BoT, it asks the Oracle when to start Cloud workers to accelerate the BoT execution.

- When needed, 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 to 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 an 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 $d = max(6000, 6000 - user\_credit\_spent)$ credits into an account. Furthermore, the mechanism allows one to easily implement more complex policies, such as the "network of favors"[**?**], 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 $\pm$ 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 (`D`): 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 $var(x) = t_c(x) - t_a(x)$. 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 $var(c) \geq 2.max_{x \in [0,50\%]}(var(x))$.

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 be completed.

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 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.

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.

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
```

3.7 Implementation

SpeQuloS has been developed as a set of independent modules, using the Python programming language and MySQL databases. Each module can be deployed on different networks (for instance, across firewalls), and communication between modules use web services. 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 different teams of the EDGI consortium. The SpeQuloS source code is publicly available[1].

Desktop Grids Middleware and Grids 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

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

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[?] developed by SZTAKI is used in the EDGI infrastructure to provide Grid and Desktop Grid interoperability. Tasks submitted to a regular Grid's computing element connected to the 3G-Bridge may be transparently redirected to a Desktop Grid. To enable SpeQuloS's 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.

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 implement 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[?] as an IaaS cloud.

4 Evaluation

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

To obtain synthetic traces of BoT execution, we have developed simulators of BOINC and XWHEP, which can optionally simulate SpeQuloS utilization.

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 datasets are provided by the Failure Trace Archive [?]. However, to our knowledge, there was no availability measurement for Cloud Spot instances or Grid systems used in best effort mode. As summarized in Table 2, we collected following traces:

- *Desktop Grid:* For this study we consider the public volunteer computing project SETI@Home (`seti`) ran by BOINC[?], and the private Desktop Grid deployments at University Notre Dame, ran by Condor[?] (`nd`). All these traces are provided by the Failure Trace Archive[?].
- *Best Effort Grid:* We consider using best effort queues of Grid5000[?] (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 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 an 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.

Table 2: Summary of the Best Effort DCI traces. The trace length, number of nodes average (*mean*), standard deviation (*std. dev.*), minimum (*min*) and maximum (*max*) are presented. *av. quartiles* and *unav. quartiles* are the nodes availability and unavailability duration quartiles, in seconds. *avg. power* and *power std. dev.* are the average node power (in instructions per second) and node power standard deviation.

trace	length (days)	mean	dev.	min	max	av. quartiles (s)	unav. quartiles (s)	avg. power (nops/s)	power std. dev.
seti	120	24391	6793	15868	31092	61,531,5407	174,501,3078	1000	250
nd	413.87	180	4.129	77	501	952,3840,26562	640,960,1920	1000	250
g5klyo	31	90.573	105.4	6	226	21,51,63	191,236,480	3000	0
g5kgre	31	474.69	178.7	184	591	5,182,11268	23,547,6891	3000	0
spot10	90	82.186	3.814	29	87	4415,5432,17109	4162,5034,9976	3000	300
spot100	90	823.95	4.945	196	877	1063,5566,22490	383,1906,10274	3000	300

Table 3: 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	$norm(\mu = 1000, \sigma^2 = 200)$	$norm(\mu = 60000, \sigma^2 = 10000)$	$weib(\lambda = 91.98, k = 0.57)$

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 are 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.

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[?, ?] allow us to model nodes power. Table 2 shows BE-DCIs workers computing power drawn from that 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.

4.1.2 BoT Workloads

BoT applications are a major source of DCIs workload. We follow the definition of BoT given in [?, ?] where a BoT is an ordered set of n independent tasks: $\beta = \{T_1, ..., T_n\}$. 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 $AT(T_i) < AT(T_j)$ id $i < j$. More, we define ϵ, the maximal time between two tasks arrivals, thus we have $\forall i \; in(1, n), AT(T_{i+1}) - AT(T_i) < \epsilon$. A typical ϵ value is 60 seconds, as used [?].

BoTs are also defined by there *size* i.e. the number of tasks. Each task also have 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.

We used 3 BoT categories in our experimentation, called `SMALL`, `RANDOM` and `BIG`. Those BoTs vary in terms of size, number of instructions per task and task arrival times. Table 3 summarizes the BoT attributes which were empirically generated based on our experience with the EDGI infrastructure and values observed in [?]. As shown in the table, `SMALL` and `LARGE` BoTs are homogeneous BoT, whereas `RANDOM` attributes were statistically generated, following the BoT analysis performed by [?].

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.

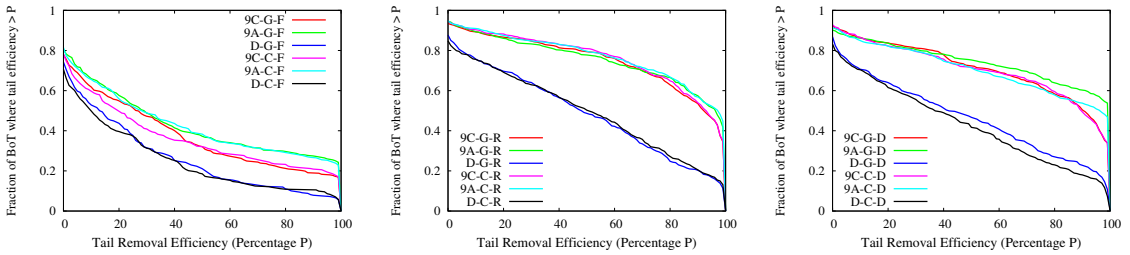
Pseudorandom number generator used in simulators can be initialized by a seed value, to reproduce exactly the same simulation executions. Therefore, using the same seed value allows a fair comparison between a BoT execution where SpeQuloS is used and the same execution without SpeQuloS.

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 given by 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 3 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 Section 3.5. We evaluate every combination 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 the tasks which belong to the tail are all duplicated to the Cloud (Cloud Duplication).



(a) Flat deployment strategy (b) Reschedule deployment strategy (c) Cloud duplication deployment strategy

Figure 4: 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.

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 $TRE = 1 - \frac{t_{speq} - t_{ideal}}{t_{nospeq} - t_{ideal}}$, 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 4a, 4b and 4c 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. 4c, `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, half of the BoT executions have an efficiency not higher than 30%, regardless of the other strategies used. For Reschedule and Cloud Duplication strategies, only the worst 40% of executions have a similar efficiency and it is even 20% of execution if the Execution Variance is excluded. Figures 4b and 4c also show that any combination of Completion threshold and Assignment threshold strategies as well as Greedy and Conservative strategies obtain the best efficiency. 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 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 criteria for the performance comparison of the strategies is the Cloud resource consumption. Lower is the resource consumption, better is the strategy. As in our system, 1

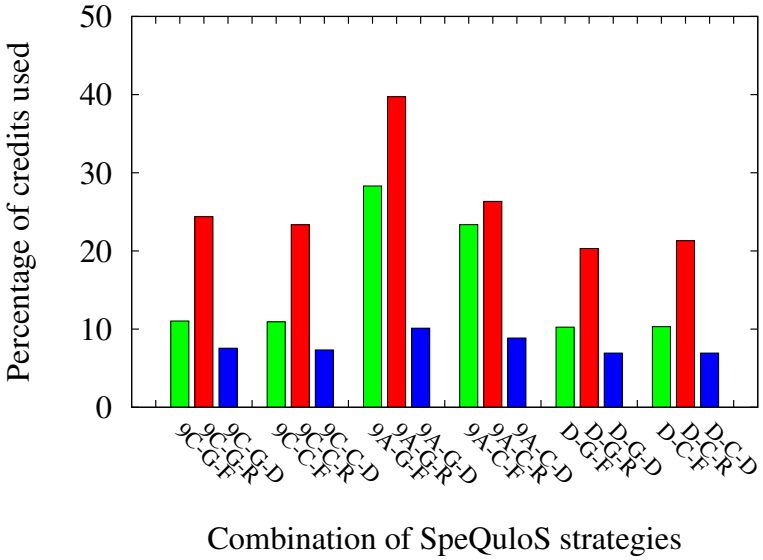


Figure 5: Credits consumption of various SpeQuloS strategies combinations. Lower is better.

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 5 shows the average percentage of credits spent against the credits provisioned. In most of cases, less than 25% of provisioned credits are spent. As 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 5 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 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.

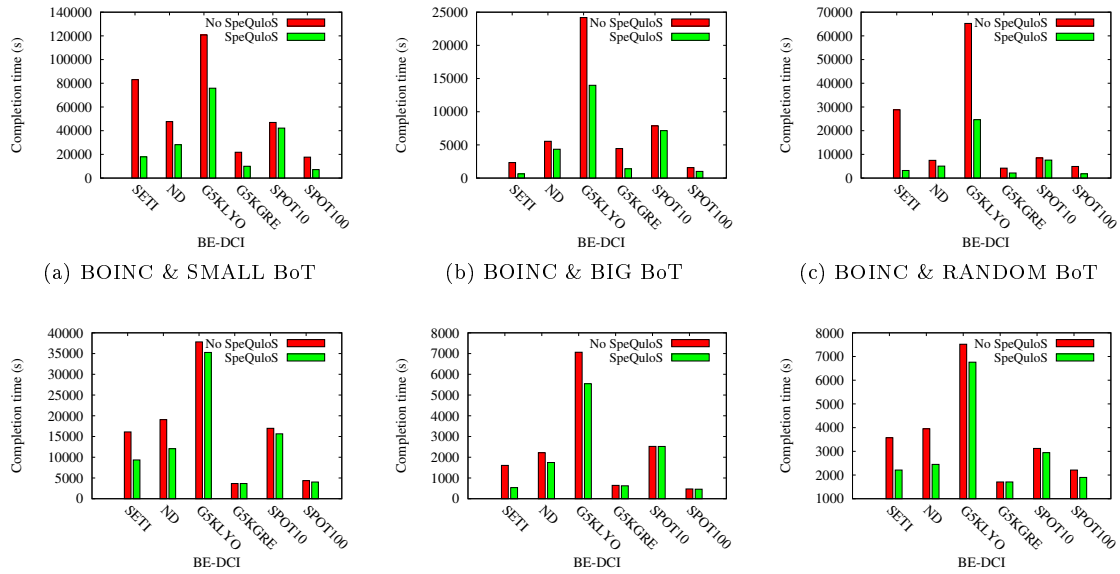
Overall, our strategies have a low credit consumption. It ensures that enough credits are supplied to support the BoT execution until it ends and leaves more credits to users to support other BoT executions.

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 6a, 6b, 6c, 6d, 6e and 6f show the average BoT completion time measured with and without SpeQuloS. Each figure presents results from one DG middleware and BoT. Each figure's pair of columns show results for each BE-DCI trace.



(a) BOINC & SMALL BoT (b) BOINC & BIG BoT (c) BOINC & RANDOM BoT

(d) XWHEP & SMALL BoT (e) XWHEP & BIG BoT (f) XWHEP & RANDOM BoT

Figure 6: Average completion time measured with and without SpeQuloS under various execution environments.

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 `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).

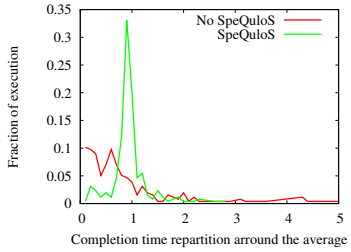
More important benefits are observed with highly volatile BE-DCIs (`seti`, `nd`, `g5klyo`). As the tail effect is more important in these BE-DCIs, using SpeQuloS can significantly increase the performance.

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.

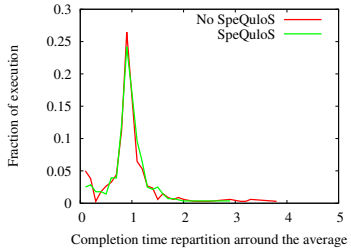
Even if BOINC and XWHEP completion times cannot be compared, as BOINC uses tasks replication while XWHEP does not, and as these middleware differ in the way they handle node failures, note that XWHEP is slightly less improved than BOINC when SpeQuloS is used.

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 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 7a and 7b show the repartition functions of normalized BoT completion times around the average. Each execution's 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



(a) BOINC



(b) XWHEP

Figure 7: 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.

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, Figure 7a 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 a very satisfactory execution stability, actually better than for XWHEP.

4.3.3 Completion Time Prediction

Table 4 shows the percentage of successful SpeQuloS predictions, described in Section 3.4, made when the BoT completion is 50%. A successful prediction is reported when the actual completion time fits the SpeQuloS predicted time associated with an uncertainty of $\pm$ 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 $\pm$ 20% in 9 cases out of 10, which is remarkable given the unpredictable nature of BE-DCIs. Results also show that predictions performed 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 inferior prediction quality. Indeed, as this BoT is highly heterogeneous, predicting completion time is harder as task execution times

Table 4: 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.

	BoT category & Middleware						
	SMALL		BIG		RANDOM		
BE-DCI	BOINC	XWHEP	BOINC	XWHEP	BOINC	XWHEP	**Mixed**
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**

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 Deployment in the European Desktop Grid Infrastructure

In this section, we present the deployment of SpeQuloS as a part of the European Desktop Grid Infrastructure[?] (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, eScience portal and more.

We present the current preliminary deployment of SpeQuloS, on part of the EDGI production infrastructure, which is illustrated in Figure 8. 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 Grid'5000 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's 3G Bridge[?]. 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 is in progress, to provide QoS support to other EDGI's DGs, such 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 5

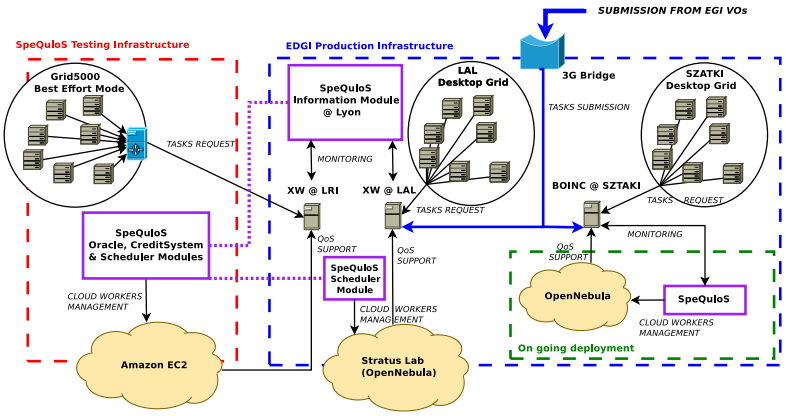


Figure 8: SpeQuloS' current deployment as a part of the EDGI infrastructure. SpeQuloS' modules are split and duplicated across the deployment.

Table 5: 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

summarizes the usage of the infrastructure during the first half of 2011 where SpeQuloS has been gradually deployed.

6 Related Work

Many scenarios motivate the assemblage of Grids, Clouds with Best Effort infrastructures, and in particular Desktop Grids. GridBot [?] puts together Superlink@Technion, Condor pools and Grid resources to execute both throughput and fast-turnaround oriented BoTs. The European FP7 projects EDGeS[?] and EDGI[?], 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 [?]. In [?], authors investigate the cost and performance of running a Grid workload on Amazon EC2 Cloud. Similarly, in [?], the authors introduce a cost-benefit analysis to compare Desktop Grids and Amazon EC2. ElasticSite[?] offloads a part of the Grid workload to the Cloud when there is peak user demand. In [?], authors propose a Pareto efficient strategy to offload Grid BoTs whith deadlines on the Cloud. One can imagine other combinations within hybrid DCIs. For instance, CATCH [?] uses the Cloud storage service to improve data access between Desktop worker and HPC centers. Our work focuses on a particular usage where most of the computing resources are provided by Best Effort DCIs and only a critical fraction of the resources are provisioned by public or private Clouds.

Providing QoS features in Grids is hard and not solved yet satisfactorily [?, ?, ?]. It is even more difficult in an environment where there are no guaranteed resources [?]. 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 [?], a framework is presented which would be the closest work to ours. Similarly, Aneka [?] support the integration between Desktop Grids and Clouds although we went further in term of implementation and evaluation.

There exists a large literature about predicting tasks completion time. For instance QBETS [?] uses time series to model and forecast task queues. Closer to our context [?], 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 an 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 [?]. 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 [?], 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[?] 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.

7 Conclusion and Future Works

Although Best Effort Distributed Computing Infrastructures (BE-DCIs) such as Desktop Grids, Best Effort Grids or Cloud Spot instances are the "low cost" solution available to high-throughput computing users, they are now getting more widely accessible. We have introduced SpeQuloS, a framework to enhance QoS for BoT applications when executed in BE-DCIs. We hope that this effort will help to make BE-DCIs "first class citizens" in the computing landscape.

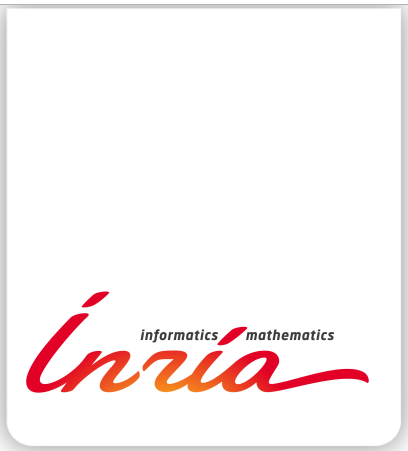
The main principle of SpeQuloS is to monitor the execution of BoTs and dynamically provision external stable and powerful Cloud resources to help BE-DCIs to execute the most critical part of the BoT. We proposed several strategies and evaluated them using trace-driven simulations. Providing QoS to grid computing is considered a difficult issue, however our approach is able to substantially improve QoS with respect to several criteria, namely completion time, completion time stability and prediction, and just as important, feedback to the user on the predicted QoS benefits.

Development and deployment of SpeQuloS have shown the potential but also the difficulties of mixing hybrid infrastructures. Our framework is composed of several small independent and distributed modules which accomplish several key tasks: information retrieval and archiving, accounting and arbitration, prediction and forecasting, scheduling and resource provisioning. We have demonstrated its applicability to the European Desktop Grid Infrastructure, where the service provides QoS support for two Desktop Grids and one Best-effort Grid connected to three different Clouds. We are now working to integrate the system into the project's Grid portal so that end-users can benefit from the service, and we hope to significantly improve their experience of using the infrastructure.

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.

Acknowledgment

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.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399