Intermediate QoS Prototype for the EDGI Infrastructure

Simon Delamare, Gilles Fedak, Derrick Kondo, Oleg Lodygensky, Péter Kacsuk, Jozsef Kovacs, Filipe Araujo

Intermediate QoS Prototype for the EDGI Infrastructure

Simon Delamare, Gilles Fedak, Derrick Kondo, Oleg Lodygensky, Peter Kacsuk, Jozsef Kovacs, Filipe Araujo

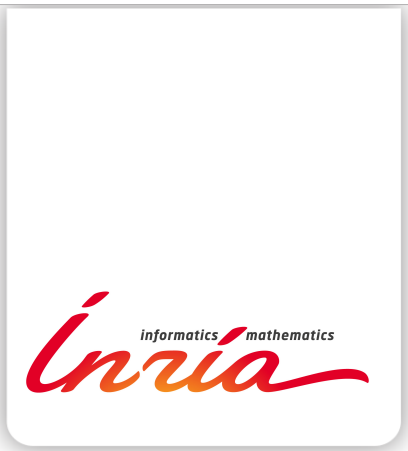
RESEARCH REPORT

N° 8294

April 2013

Project-Team GRAAL

ISSN 0249-6399 ISRN INRIA/RR--8294--FR+ENG



Intermediate QoS Prototype for the EDGI Infrastructure

Simon Delamare*, Gilles Fedak†, Derrick Kondo‡, Oleg Lodygensky§, Peter Kacsuk¶, Jozsef Kovacs‖, Filipe Araujo**

Project-Team GRAAL

Research Report n° 8294 — April 2013 — 33 pages

Abstract:
This document provides the first deliverable of EDGI JRA2. It is produced by the INRIA team, the SZTAKI team, the LAL/IN2P3 team and the University of Coimbra team. This document aims at describing achievements and results of JRA2 tasks "Advanced QoS Scheduler and Oracle" and "Support In Science Gateway".
Hybrid Distributed Computing Infrastructures (DCIs) allow users to combine Grids, Desktop Grids, Clouds, etc. to obtain for their users large computing capabilities. The EDGI infrastructure belongs to this kind of DCIs. The document presents the SpeQuloS framework to provide quality of service (QoS) for application executed on the EDGI infrastructure. It also introduces EDGI QoS portal, an user-friendly and integrated access to QoS features for users of EDGI infrastructure.
In this document, we first introduce new results from JRA2.1 task, which collected and analyzed batch execution on Desktop Grid. Then, we present the advanced Cloud Scheduling and Oracle strategies designed inside the SpeQuloS framework (task JRA2.2). We demonstrate efficiency of these strategies using performance evaluation carried out with simulations. Next, we introduce Credit System architecture and QoS user portal as part of the JRA2 Support In Science Gateway (task JRA2.3). Finally, we conclude and provide references to JRA2 production.

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
¶ MTA/STZAKI - peter.kacsuk@sztaki.mta.hu
‖ MTA/STZAKI - kovacs.jozsef@sztaki.mta.hu
** University of Coimbra - filipius@dei.uc.pt

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

Prototype intermédiaire assurant la Qualité de Service à l'infrastructure EDGI

Résumé : Ce document fournit le premier livrable pour la tâche JRA2 du projet européen European Desktop Grid Initiative (FP7 EDGI). Il est produit par les équipes de l'INRIA, de SZTAKI, du LAL/IN2P3 et de l'Université de Coimbra. Ce document vise à décrire les réalisations et les résultats qui concernent la qualité de service pour l'infrastructure de grilles de PCs européenne EDGI.

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

Project number: RI 261556

Project acronym: EDGI

Project full title: European Desktop Grid Initiative

Research Infrastructures
INFRA-2010-1.2.1 Distributed computing infrastructures
DCI-1.2.1.3 Middleware and repositories

D7.1: Intermediate QoS prototype for EDGI infrastructure

Due date of deliverable: 31/03/2011 Actual submission date: 31/03/2011

Start date of project: 01/06/2011 Duration: 24 months

INRIA

Dissemination level: PU

Version: 1.2

EDGI is supported by the FP7 Capacities Programme under contract nr RI 261556

Contents

1 Glossary

BOINC	Berkeley Open Infrastructure for Network Computing
BoT	Bag of Task
DG	Desktop Grids: Grids composed of Desktop PCs.
EDGeS	Enabling Desktop Grids for E-Science, the FP7 project that aims to bridge Service Grids and Desktop Grids
EGEE	The Enabling Grids for E-sciencE (EGEE) project is funded by the European Commission and aims to build on recent advances in grid technology
FTP	File Transfer Protocol
INRIA	Institut National de Recherche en Informatique et en Automatique (Saclay, France)
JSDL	Job Submission Description Language (OGF recommendation)
LAL	Laboratoire de l'accelerateur Lineaire (Orsay, France) belonging to IN2P3
LCG-CE	LCG Computing Element
LRI	Laboratoire de Recherche en Informatique (Orsay, France)
QoS	Quality of Service
SG	Service Grids: Grids composed of resources belonging to institutions.
SpeQuloS	JRA2 software to support QoS in Desktop Grids
SQL	Structured Query Language
VO	Virtual Organization
UI	User Interface machine (EGEE)
VOMS	VO Management Service
WMS	Workload Management System (EGEE)
WN	Worker Node
WU	Workunit
XW	XtremWeb
XWHEP	XtremWeb for High Energy Physics: Grid middleware developed by IN2P3 LAL (from XtremWeb of INRIA) and used inside the EDGI project

2 About this document

This document provides the first deliverable of EDGI JRA2. It is produced by the INRIA team, the LAL/IN2P3 team, the University of Coimbra team and the SZTAKI team. This document aims at describing the **SpeQuloS** prototype to provide quality of service (QoS) for application executed on the EDGI infrastructure.

Hybrid Distributed Computing Infrastructures (DCIs) allow users to combine Grids, Desktop Grids, Clouds, etc. to obtain for their users large computing capabilities. The EDGI infrastructure belongs to this kind of DCIs. In this document, we present the architecture, design and implementation of the SpeQuloS framework which aims at providing QoS to Desktop Grid by provisioning on-demand Cloud resources. We also report on real usage of the framework and describe evaluation on the experimental platform which relies on Grid5000 and Amazon EC2 to mimic both a Desktop Grid system and a Cloud system. Preliminary results are presented which shows the potential of the SpeQuloS approach.

3 Introduction

There is a growing demand for computing power from scientific communities to run their applications and process large volumes of scientific data. Meanwhile, the supply of Distributed Computing Infrastructures (DCIs) for scientific computing continues 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.

As an example of such hybrid DCIs, the European FP7 projects EDGeS[1, 2, 3, 4, 5, 6, 7] and EDGI[8] have developed bridge technologies to make Desktop Grid (DG) systems, such as BOINC[9] or XtremWeb-HEP[10, 11, 12, 13, 14] (XWHEP) transparently available to any Enabling Grids for E-sciencE[15] (EGEE) grid users as a regular cluster. Indeed, other attempts have successfully built systems where Grid infrastructures are supplemented by low cost volunteer computers[16] or by Cloud resources[17], benefiting from the elastic resource provisioning, to meet users' peak demands.

The main problem with the current EDGI infrastructure is that it cannot give any QoS support for those applications that require a faster execution in the public DG part of the infrastructure. For example, a public DG system enables clients to return work-unit results in the range of weeks. There is a clear need for the infrastructure to support applications which require a lower latency.

We define quality of service concretely as a probabilistic guarantee of job makespan or throughput. Job makespan is the time between the job is first submitted and the time all the results have been received and validated. Throughput is the rate at which tasks in the job are completed. Providing QoS features even in Service Grids is hard and not solved yet satisfactorily. It is even more difficult in an environment where there are no guaranteed resources. In DG systems resources can leave the system at any time for a long time or forever even after taking several work-units with the promise of computing them.

The approach followed by JRA2 work package to improve QoS management is to develop the SpeQuloS middleware for the EDGI infrastructure which is composed of four main components: QoS Info, QoS Credit System, QoS Scheduler and a QoS Oracle. These components will be responsible to decide on allocating additional Cloud resources for a certain application if QoS requirements need it and the user has already collected the necessary number of credits by supporting the public DGs of the EDGI infrastructure with desktop resources. The monitoring system will be developed in order to store QoS information it will provide a repository of data on DGs' infrastructure workloads.

At this stage of the project, we have developed prototypes of the Info, Credits, Oracle and Scheduler. SpeQuloS features multi-DG support (BOINC, XWHEP), multi Cloud services support (Eucalyptus, Amazon EC2, Rackspace, G5K). In this report, we give a detailed view of each SpeQuloS components as well as their implementation. SpeQuloS itself is a distributed architecture which relies on Web service protocols for communication between the components. We present basic usage and use case. Next, we describe our large scale testbed, called XtremG5K, which links Desktop Grid and experimental Grid (Grid5000) through an XWHEP server and we report on preliminary experimental results. Finally, we describe the Grid5000 libcloud[18] driver that we developed to start VM instances on Grid5000 as with any commercial Clouds.

4 SpeQuloS Principle and Architecture

SpeQuloS is a framework to provide Quality of Service (QoS) to users of unreliable distributed computing infrastructures (DCI), such as Desktop Grids (DG) based on volunteer computing,

by provisioning stable resources from Cloud services.

Scientific workload are usually diverse, consisting of long high-throughput jobs and relatively short-lived jobs. Users submitting high-throughput jobs only care about the rate at which job tasks are completed. The volatility and heterogeneity of individual desktop resources is of less a concern compared to the overall aggregate computing power. In contrast, users submitting short-lived jobs consisting of batches of tasks often require the batches to be completed within some time frame. Typically, users submit batches with tens of thousands of tasks and require completion within an order of 10 days. Here, jobs must be scheduled on reliable and fast resources.

Our focus in JRA2 is on how to support QoS for jobs needing fast turnaround time, as this is more required by SG users than guaranteeing throughput. As a consequence, QoS is described by Bag of Tasks (BoT) response time, which is the time from BoT submission to results delivery. In particular, we aim at reducing the tail effect, as illustrated by Figure 1. The tail is the last fraction of the remaining tasks to execute which is the causes for the BoT execution slowdown. The tail is mostly due to host volatility, and the time needed by the DG middleware to detect host or tasks failure and reschedule the tasks lost. XtremWeb uses a host failure detector based on heartbeat and BOINC uses task deadline and task replication. In both cases, at the end of the BoT execution, the tasks which were assigned to a host which have left the system have to be assigned to some other nodes, which causes a drop in the job completion rate. We define as a BoT a collection of work units submitted by user to a DCI. The main objective of SpeQuloS is to bring to users some information on their BoT response time and to decrease this time by provisioning resources from Cloud services if they so requested.

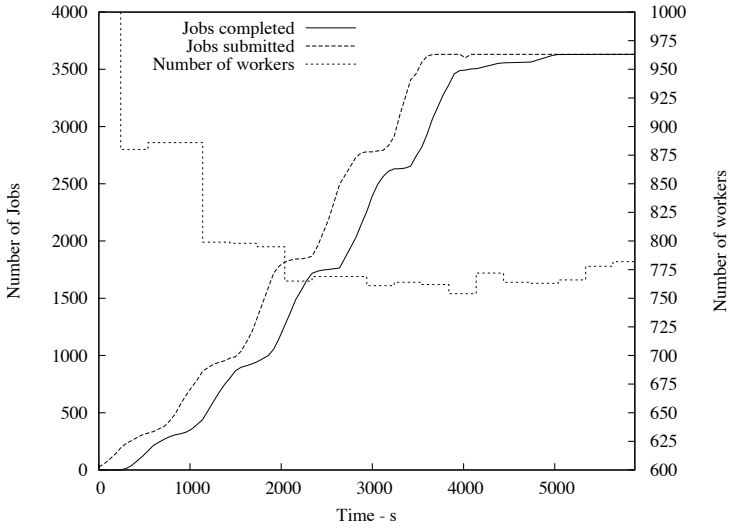


Figure 1: BoT execution with a tail

To decrease a BoT response time, SpeQuloS adds dedicated resources to its computation. SpeQuloS uses Infrastructure as a Service (IaaS) Clouds to provide those resources. In the scope of the EDGI project, it is anticipated that Cloud resources will be provided by partners, thus we do not consider commercial Cloud infrastructure as a final target. Instead, we assume that Cloud system will be private, i.e. accessed only by the consortium members to help EDGI DGs.

To supply resources to a DG, SpeQuloS instantiates a virtual machine, called Cloud worker, starts some DG worker software, makes it join the DG server where the BoT is computed and stops it when necessary. As access to Cloud resources is limited, SpeQuloS provides a framework

to share those resources between users and account their utilization. To use Cloud resources in a smart way, SpeQuloS gathers information about the QoS in a DG. This information is used to evaluate the current level of QoS and to anticipate benefits of allocating cloud resources.

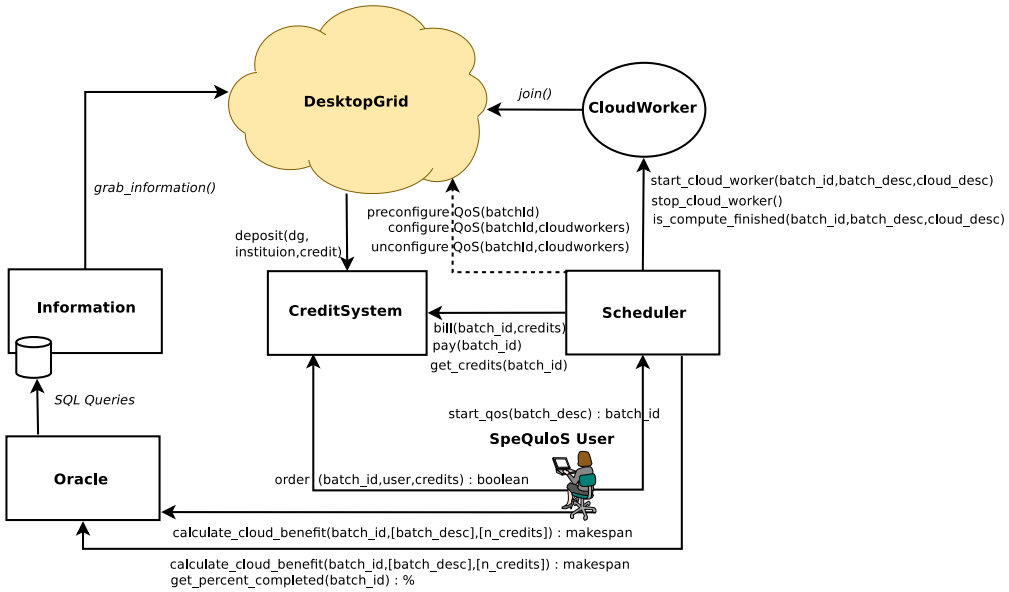


Figure 2: The SpeQuloS modules and their interactions

SpeQuloS is composed of several modules as shown in Figure 2:

- The Credit System module is in charge of accounting the Cloud-related operations. It interacts with users to receive orders to provision a BoT with Cloud resources and with the Scheduler module to bill their utilization.
- The Information module gathers and stores information relevant to QoS from DG, such as the BoT completion (which denotes the ratio between the number of completed and uncompleted jobs) and the number of workers participating in the DG.
- The Oracle module makes some prediction about QoS in the DG. Using the information provided by the Information module, the Oracle computes an estimated response time for a BoT. The response time is estimated with and without use of Cloud resources, to point out its benefit. The main role of the Oracle module is to give information to users who take decision to spend Cloud resource for their BoT.
- The Scheduler module manages the BoT and the Cloud workers during the system execution. When QoS are requested by an user, it instantiates some Cloud workers and assigns them to the appropriate BoT. The module declares to the Credit System the Cloud resources spent. It also interacts with the DG server, to perform some configuration related to BoT and Cloud workers, which depends on the type of DG considered.

5 Usage and Use Case

Figure 3 presents a simplified sequence diagram of the usage of SpeQulos and the different interactions between the components of the system. The detailed "step-by-step" use case with

corresponding screen capture can be found in Annex A.6.

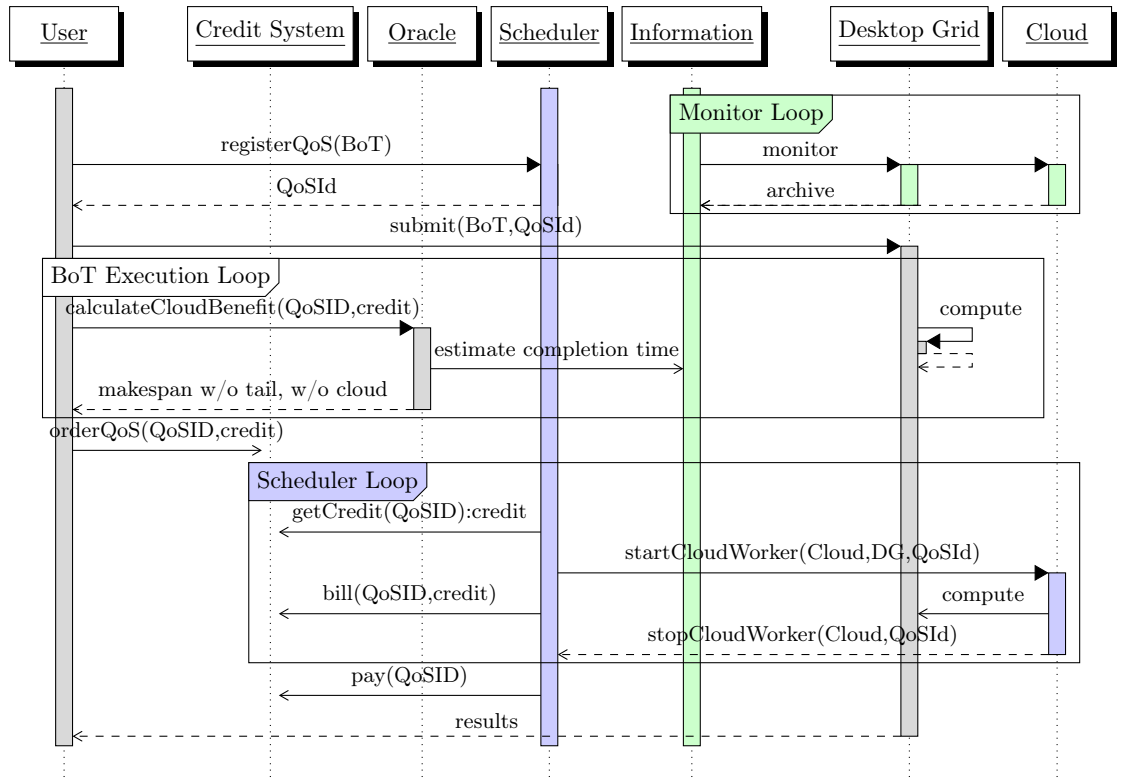


Figure 3: SpeQuloS sequence diagram

The progression of the scenario is represented vertically, and the various function calls between SpeQulos modules are represented by arrows. Following is a description of the various steps of this scenario:

- The first step of the scenario is the request for QoS by a user, for one of its BoT, to the Scheduler. The scheduler returns to the user a BoT identifier (batch id) that must be used for the other transactions related to this BoT. The scheduler also adds the BoT to the list of its monitored BoTs. The scheduler may have to pre-configure the targeted DG to accept QoS. Then, the user submits its BoT to the DG, as he normally does, except that he needs to use the batch id given by the scheduler.
- At any moment, the user can request the oracle to calculate the benefits of using cloud resources to improve the QoS of his BoT. According to the oracle answer, the user can order some QoS to the credit system. By doing this, he declares to the credit system that he is ready to spend a number of credits to improve the QoS of the BoT. The credit system then verifies that there is enough credits on the user's institution account to allow the order, and then allocate the credits to the user's BoT.
- The scheduler periodically asks the credit systems if there is some credits allocated for the BoTs it manages. If some credits are available, and if the BoT completion is advanced enough, the scheduler asks the oracle what benefits for BoT's QoS can be expected by using cloud resources.

- If it worth to spend those credits, the scheduler starts a cloud worker. The cloud worker joins the DG, as any else worker would do. The scheduler then configures the DG to ensure that the cloud worker will only compute the jobs associated to the BoT for which credits have been spent.
- 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 spent. If all the credits allocated to the BoT have been spent, or if the BoT computation finished, the cloud worker must be stopped, and the cloud worker assignment to BoT must be unconfigured in the DG.
- The scheduler then asks to the credit system to pay for the resources usage. The credit system then closes the order relative to the BoT. If the BoT computation was completed before all the credits have been spent, the credit system transfer back remaining credits to the user's institution account.

6 QoS Components

In this section, each module features will be presented in details. However, database implementation and programming interface details are not covered in this document. We orient the reader towards the SpeQuloS documentation, given as Annexe A, which documents all the functions of the middleware as well as database schema and entity-relation description.

6.1 QoS Credit System

The credit system module has the following responsibilities: 1/ Usage Accounting and 2/ Cloud Usage Arbitration.

Interface is similar to banking system where virtual credits are accounted (deposit, bill, pay,...). The credit system stores the information about the credit accounts for the institutions, clouds, DG and orders given by users for their BoTs. At the moment we are using a flat correspondence between Cloud worker usage and credits (1 CPU.hour = 15 credits).

The credit system interacts with the users to receive orders, and with the scheduler to exchange information about credit spent for BoTs by spending some cloud resources. It also receives some deposit requests from DG administrators to fund an institution account.

The Credit system's database includes the following tables:

- QoSUser: Stores the credits spent by each users participating in SpeQulos, for statistics.
- Institution: Stores the credits balance for each institution, as well as the credits already won and spent, for statistics.
- DesktopGrid: Stores the credits balance for each DG, as well as the credits already won and spent, for statistics.
- Cloud: Stores the credits spent by each clouds participating in SpeQulos, for statistics.
- QoSOrder: Stores the credit balance for each order created by user for his BoT, as well as the credit originally funded.

The policy for credit deposit is given and implemented by Desktop Grid administrator. Here we give two basic examples of deposit policies:

- *local policy:*
  - Grid5K has a usage policy where one user should not reserve the whole platform for more than 2 hours.
  - SpeQulos is not authorized to reserve more than 200 nodes per day
  - the user deposit function run every 24 hours is $d = max(6000, 6000 - user\_credit)$
- *public policy:*
  - Volunteer projects such as BOINC maintain a credit accounting to reward volunteers participation.
  - Allows volunteers to support a project twice: by providing their CPU and by "donating" their credit to an institution
  - the user deposit function run every 24 hours is $d = volunteer\_credit/\#users$

6.2 QoS Information

The information module collects and archive the information on DG activities relevant to implement QoS strategies. The QoS information has two responsibilities: 1/ provide real time information on BoT execution and DG computational activities and 2/ archive BoT execution traces from which statistical model can be extracted in order to make BoT execution prediction. QoS Info provides these information to the QoS Oracle.

One of the key point of QoS Info is to hide Desktop Grid idiosyncrasies, i.e. different Desktop Grid technologies which have specific way of managing queues should appear in an unified format. Currently, QoS Info stores the information of batches completion history (in terms of number of completed jobs of a batch, compared to number of uncompleted jobs, as a function of time) in the following format:

```
timestamp DG #worker #cloudworker
timestamp BoT #job_completed #job_uncompleted #job_assigned
```

Figure 4 shows the implementation of the information module. First an agent called the *DG wrapper* has to be installed on Desktop Grid server. To facilitate its deployment, the wrapper is provided as a PHP extension to the DG server web site. The QoS info periodically poll the wrappers and stores the monitored information in two database tables.

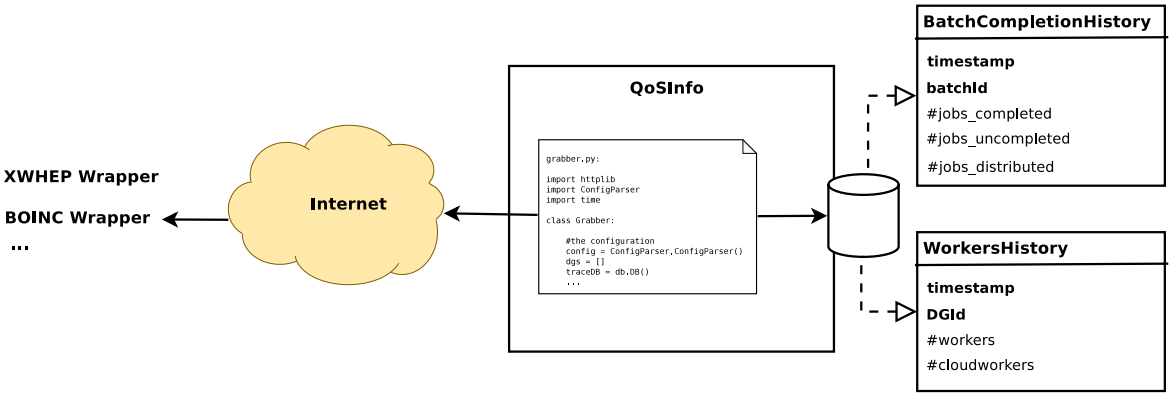


Figure 4: QoS Info architecture

During the development of SpeQuloS, we experimented with several format for DG activity representation and archiving. Now that the information format is stabilized, QoS Info should be merged with the monitoring task of JRA1.

6.3 QoS Oracle

The QoS Oracle has the responsibility of deciding when and how many Cloud resources should be provided to the Desktop Grid to ensure BoT QoS. Using the information provided by the Info module, the oracle's module computes an estimated makespan for a batch. The makespan estimations are performed with and without the use of cloud resources, to emphasize the benefit of using the cloud. The oracle module interacts with the users to help them to decide if it worth to spend credits for their batches, and with the scheduler, which verifies if it is really useful to use some cloud resources for a batch which has some credits allocated to it.

The Oracle strategy, called `Calculate_ cloud_ benefit` to trigger the QoS is the following:

- Let $T_e(x)$ be the elapsed time from BoT submission when x is the fraction of the BoT which is completed. $\frac{T_e(x)}{x}$ is the BoT completion rate.
- At time t, x_t and $T_e(x_t)$ are given by the monitoring.
- If we assume the completion rate is constant, we have

$$T_{notail} = T_e(1) = 1.\frac{T_e(x_t)}{x_t}$$

- Let x_{tail} the fraction of the BoT affected by the tail and σ the slowdown factor when the executing the tail. We have

$$T_{tail} = T_e(1 - x_{taill}) + \sigma.T_e(x_{tail})$$

$$T_{tail} = (1 - x_{taill}).\frac{T_e(x_t)}{x_t} + \sigma.x_{tail}.\frac{T_e(x_t)}{x_t}$$

Let T_{cloud} be the estimated time using Cloud workers. If Cloud workers are deployed to compute the last x_t fraction of the BoT completion and if ρ be the ratio of DG worker vs Cloud performance, we have:

$$T_{cloud} = (1 - x_{taill}).\frac{T_e(x_t)}{x_t} + \rho.x_{tail}.\frac{T_e(x_t)}{x_t}$$

Figure 5 shows the curves presenting the expected completion time, with and without tail and the advantage of using Cloud resources. Here, we consider the following parameters. First, we assume that the fraction of tasks in the tail is fixed proportion of the BoT: $x_{tail} = 10\%$, we assume that once in the tail, the job completion rate decreases by a factor 2, i.e. $\sigma = 2$ and that the ratio between the Cloud worker performance and the average DG worker performance, including the slowdown due to its volatility is $\rho = 0.33$, which means that Cloud workers completes 3 times more work units than DG worker. The former parameters has been established by prior studies found in the literature [19] comparing EC2 with SETI@Home.

At any time, EDGI user can monitor their BoT execution and query the Oracle to obtain BoT prediction time (see Figure 6). User can then decide to support their BoT by allowing credit to SpeQuloS.

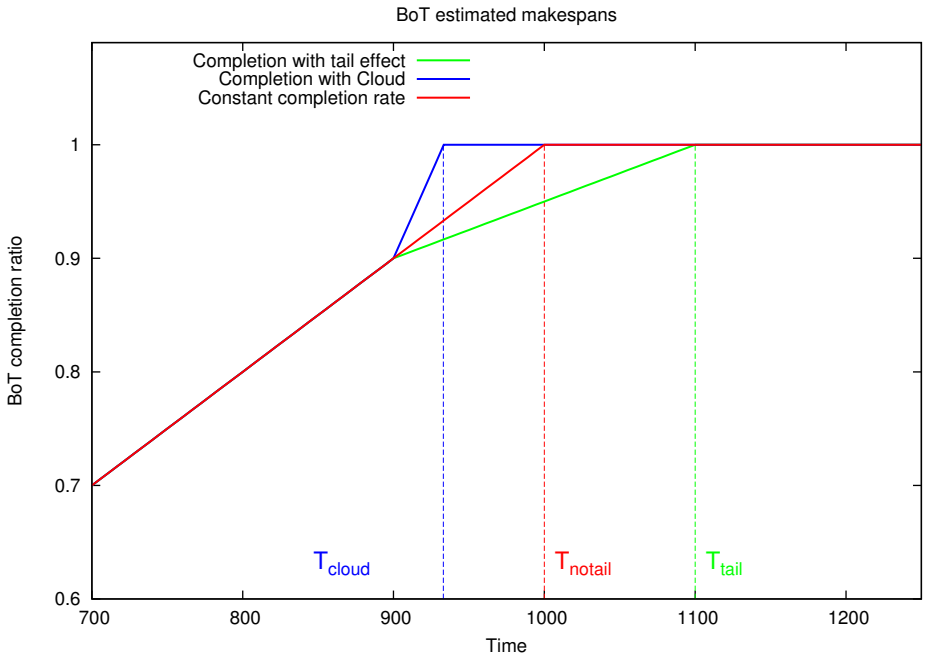


Figure 5: Cloud benefit strategy

During the execution of the tail, the scheduler uses real-time estimation T_{tail} and T_{cloud} to make the decision of using Cloud resources. The decision of provisioning Cloud resources is made if:

$$T_{cloud} < T_{tail} \cdot \gamma$$

where γ is the benefit factor of using the Cloud (here γ is fixed to 0.75 in our implementation). If the decision is made, then additional Cloud workers will be provisioned to compute the remaining 10% of the BoT.

However, one can note that the current strategy suffers from strong drawback:

- x_{tail} depends certainly on the BoT and DG characteristics: number of jobs, jobs length, variation of available workers, workers volatility and many more
- To predict T_{cloud}, we must estimate ρ, the speed-up factor of using Cloud workers against DG workers. Using a fixed ρ for the whole infrastructure is not realistic. Instead, ρ should be estimated dynamically depending on the cloud and DG characteristics.
- At the moment, SpeQuloS does not handle specifically instances of "unpredictable" batches, i.e. non time constant jobs, or when traces are not available (first run).

We anticipated this situation, and in the second year of the project, our work will focus on improving and evaluating the Oracle strategies. Also, we will add the ability for user to pre-define their preferences in terms of credits and how much they want to spend for a given benefit.

Unlike the previous two modules, the Oracle module is implemented as a stateless service which does not use a database. However, because it can make heavy read access to information stored by the Info module, we recommend that these two modules run on the same machine.

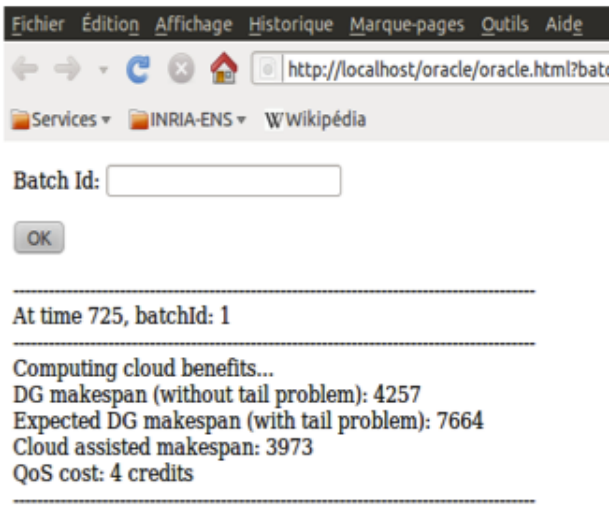


Figure 6: Oracle user interface

6.4 QoS Scheduler

The QoS Scheduler has the responsibility to provision additional Cloud resources to the execution of BoT for which users have required QoS surveillance.

Once jobs are submitted to the corresponding DG through the 3G Bridge, the QoS scheduler monitors job execution on the DG. According to user QoS ability, the QoS scheduler adds additional computing resources if it detects that the DG will not be able to complete the jobs on time. Technically, this is achieved by building Cloud instances which embed the DG workers middleware. We call these instances Cloud Worker, and we build one instance per Cloud and Desktop Grid pair.

QoS scheduler starts the instances on the Cloud using a library called `libcloud`, which allows to unify the various 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 configure the worker to connect to the specific Desktop Grid and process jobs from the specific BoT. In fact, it is important to ensure that a Cloud resources on which users are spending credits should not compute tasks belonging to another users' BoT. The DG worker can fetch job from the DG server and executes this as regular jobs. As a consequence, the Cloud resources provide additional computing elements to the DG. This mechanism is similar to the one used in the Grid and known as PilotJob. As soon as the QoS scheduler detects that no more Cloud resources are needed, it frees the Cloud pilot job by remotely shutting it down. Algorithms and presents the various algorithms which monitor BoT execution and start/stop Cloud workers.

7 Implementation of the Prototype

In this section, we report on the technologies used to implement the SpeQuloS prototype.

First, it is important to notice that each module is independent and can be deployed on different computers. Technically, modules are developed as "daemon-less" code. Periodic tasks are ran through a cron process to improve the robustness of the code with respect to memory leakage.

Algorithm 1 MONITORING BOT

```
for all B in BoTs registered do
  if Info.getCompletion(B) > 1-Oracle.x_tail then
    if T_cloud < T_tail.γ then
      if CreditSystem.getCredit(B) then
        for all CW in Cloud Workers Available do
          CW.start()
          configureQoS(B.getDG(),CW)
        end for
      end if
    end if
  end if
end for
```

Algorithm 2 MONITORING CLOUD WORKERS

```
for all CW in CWs started do
  B ← CW.getBoT()
  if (Info.getCompletion(B) = 1) or (CreditSystem.bill(B) = False) then
    CW.stop()
    if not (B.hasCW()) then
      CreditSystem.pay(B)
    end if
  end if
end for
```

Python programming language and MySQL databases are used for modules implementation. Communications between modules use web services.

The source code is fully documented, as well as the API and the database schema. We provide a complete user manual as well as short notice for installation instruction, dependencies installation instruction.

Some test units improve quality of the code by preventing bug to appear during the development process. We perform regularly scalability experiments using Grid5000 and Amazon EC2 which implies several thousands of workers. We perform stability tests by letting the modules run permanently on our server and connected XWHEP Desktop Grid installed at LRI since December 2010.

SpeQuloS does not depend on a particular DG technology. Currently, SpeQuloS supports BOINC and XWHEP DG. It does not depend on a particular Cloud service either. It uses the libcloud[18] Application Programming Interface (API) as an abstraction layer to access to various types of Cloud services transparently. Cloud services tested include Eucalyptus, Amazon EC2, Rackspace and Grid5000 and in the next future we will consider OpenNebula and Nimbus.

Several DGs and Clouds can be connected at the same time to a single SpeQuloS deployment. To be managed by SpeQuloS, each DG and Cloud service can be registered by the SpeQuloS administrator through a dedicated web page.

To support BoT execution in DG, a Cloud Worker disk image must be available to the Cloud service. We provide a document describing how to build and load this image to an Eucalyptus-based Cloud and to Amazon EC2.

At the time of the writing, SpeQuloS is version 0.2.1. It has been released to the SA2 testing

infrastructure (part of the Work Package 4) with a document describing testing guideline.

7.1 QoS Desktop Grid support

One of the limitation that we have faced during the development of SpeQuloS is the lack of support for QoS within Desktop grid middleware (BOINC and XtremWeb-HEP).

Fortunately, even if DGs do not support BoT per se, we have found concepts that allow to group collective jobs in batch. In BOINC exists the batchId, which is an integer that can be used to tag individual work units. In XW-HEP exists the notion of work group. At the beginning of the submission, user creates a work group and submit individual job within this work group. Both of these two mechanisms imply that SpeQuloS has to ask the DG server to create the BoT recipient. To do so, we install a tiny wrapper on the DG server which is remotely invoked at the BoT creation time by SpeQuloS.

The second issue relates to Cloud workers which should compute jobs for a specific BoT. As previously mentioned, it is crucial to ensure that a Cloud worker on which users is spending credits is not computing tasks belonging to another users' BoT. In BOINC, we create a Cloud user and use the scheduler's assignment mechanism so that the Cloud workers can only process jobs within BoT with Cloud resource. Unfortunately, this solution requires one to patch the BOINC server as long as a similar mechanism is not included in the mainstream BOINC distribution. With XWHEP, we have asked the developers to modify the scheduler and the worker so that the worker can be started with the batchId as an execution parameter. The scheduler will then only send jobs corresponding to the specified batchId. This modification has been released in the version 7.4.0 of XtremWeb-HEP.

7.2 Cloud support

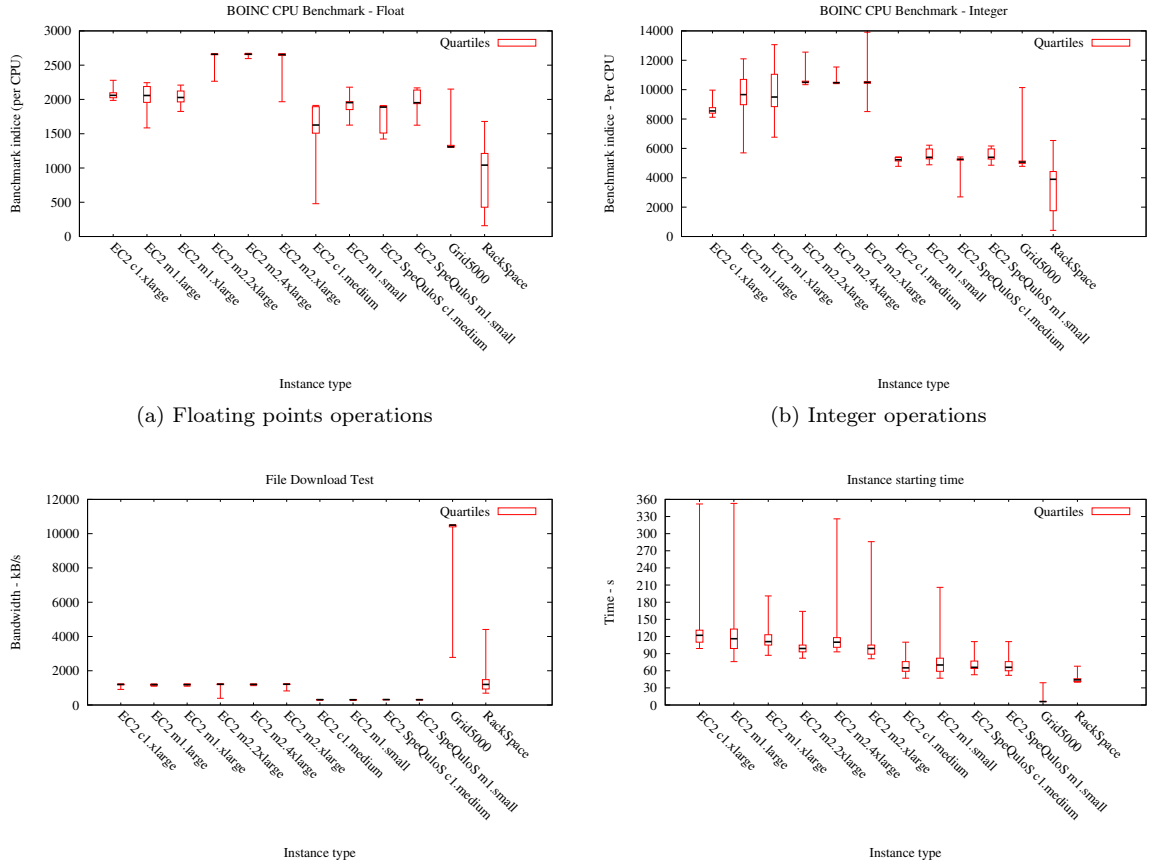
Thanks to the versatility of libcloud library, SpeQulos already supports several Cloud technologies:

- Eucalyptus and Amazon EC2. These are two compliant technologies which are deployed either on commercial Cloud or on private Cloud. We have ran extensive tests using the Amazon EC2 Clouds and we have found the service to be very reliable and efficient. Unfortunately at this moment, there is no private Cloud based on the Eucalyptus deployed on the EDGI infrastructure. Although, we have experimented using the publicly available Eucalyptus cluster opened for developers, our tests are not satisfying enough to validate the usage with Eucalyptus private Clouds.

- Rackspace is a commercial Cloud. We have successfully experienced with the Rackspace Cloud to assess the ability of our system to cope with different Cloud technologies.

- Grid5000 is an experimental Grid for research in France which can be seen as kind of private Cloud (up to 6000 cores are freely available). The difference is that Cloud deploys OS within virtual machines while Grid5K deploys OS on "bare-metal" nodes, which is transparent with respect to SpeQuloS Cloud worker implementation. We have developed a new driver for libcloud so that SpeQuloS can use G5K (see section 10.5).

Figure 7 shows the performance measurement when using several instances and Cloud technologies when running the BOINC benchmark in floating point and integer, network bandwidth measurement and the time between the start of an instance and the time the worker become available.



(a) Floating points operations

(b) Integer operations

(c) Bandwidth

(d) Start time

Figure 7: Cloud performance benchmarking

On going works are being done on supporting more Cloud technologies. First we have preliminary support for OpenNebula and Nimbus Clouds. Second, we will develop a libcloud driver for the 3G bridge, so that SpeQuloS can provision Cloud resources using the 3G bridge Cloud worker facility.

7.3 EDGI BoT submission support through SZTAKI's 3G Bridge

The 3G bridge developed by SZTAKI is used in the EDGI infrastructure to provide Grid and Desktop Grid interoperability. Jobs submitted through 3G bridge can be transparently redirected to a Grid or a Desktop Grid. SpeQuloS uses a batch identifier to identify a BoT and to trigger QoS operations. Hence, to support BoT submitted using the 3G bridge, this identifier must be propagated from the BoT description provided by user during submission, to the DG plugin of 3G bridge and then to the DG itself.

A special field, called "SPEQULOS_BATCH_ID", is used inside the 3G bridge to store the identifier used by SpeQuloS. Modifications to both BOINC and XWHEP 3G Bridge plugins have been investigated to handle this field and ensure that jobs will arrive in the DG with the

appropriate attributes.

8 SpeQuloS requirements and installation

The requirements and the procedure to install SpeQuloS are described in this section. The installation documentation provided with SpeQuloS, as well as the other available resources, are presented in Annex A.

The needed software to run SpeQuloS are

- A web server (apache version 2.2 has been successfully tested)
- MySQL server (version 5.1 has been successfully tested)
- The Python interpreter with standard environment with version >= 2.6
- The mysqldb module for Python
- The paramiko module for Python
- The libcloud module for Python (version 0.4.0 to 0.4.2 has been successfully tested)

SpeQuloS installation configuration file must be edited to, at least, provide the Mysql administrator login and password. Then, SpeQuloS can be installed using an installation script. Optionally, each of the SpeQuloS modules can be installed on different machines.

SpeQuloS has some requirement to work with desktop grids. In particular, it needs some plugin to be installed on the desktop grid servers. Those plugins are PHP web pages and must be deployed on an HTTP server on each desktop grid server.

To support an XWHEP DG, SpeQuloS requires a XWHEP server version >= 7.2.0. Some PHP web pages, provided in the SpeQuloS package, must be deployed in the XWHEP server to be accessible through HTTP. A XWHEP client must be available to this plugins.

To support a BOINC DG, SpeQuloS requires to use a patched version of the BOINC server. The patch files and the instructions are provided in the SpeQuloS package. SpeQuloS also requires to deploy some PHP web pages to be accessible through the BOINC project admin page.

The grabbers used by SpeQuloS Information module to monitor the DGs must be ran periodically on the machine hosting the Information module. For each desktop grid to monitor, a grabber program must be periodically executed by using cron for instance. To keep the information stored in the database accurate, it is advised to execute the grabber programs frequently. Scheduler module's monitoring programs must also be ran periodically by using, for instance, cron.

The desktop grid managed by SpeQuloS must be declared. Each desktop grid server must be registered through a Web service, by providing an unique desktop grid name, the desktop grid type and the URL to access to the plugins.

Each cloud resource available to the scheduler must be declared. Each cloud worker instance, with the technology of the cloud, must be registered through a Web service. The credentials to access to cloud servers must be provided in a configuration file. It is also needed to that a cloud worker image to be loaded in the server. Documentation to build and load this image is available.

Table 1: Statistics of BoT execution on EDGI's DG

	LRI DG	LAL DG
Number of BoT execution collected	451	0
Total number of individual jobs	306364	619085
Average number of workers	334	62
Maximum number of workers	1145	283
Maximum number of cloudworkers	31	0

9 Collecting and analyzing BoT execution traces on DG

Collecting BoT execution traces on DG is crucial to identify the causes of the lack of QoS offered to users. The analysis of these traces will be used to elaborate advanced QoS heuristics used in SpeQuloS. In this section, we are presenting BoT execution traces collected and analyzed by JRA2.

9.1 Monitoring EDGI's DGs

The aim of the SpeQuloS Information module is to collect the execution of BoT on DGs and store that information. A stand alone installation of this module has been set up on a machine hosted at Ecole Normale Superieure de Lyon to permanently monitor deployed DGs. Since December 2010, we are using this module to collect BoT executions on the XWHEP DG hosted at LRI. Since March 2011, the module also monitors the XWHEP server hosted at LAL. We are planning to extend this monitoring to others DGs hosted in the EDGI infrastructure, in particular those running BOINC middleware.

Table 1 shows, for each DG monitored by this module, the number of BoTs and the total number of jobs executed on the DG, the average and maximum number of workers participating to the DG and the maximum number of cloudworkers seen simultaneously.

9.2 BoT execution analysis

As number of BoT execution traces collected by the Information module was not enough to perform an exhaustive analysis of BoT execution characteristics, we used generated traces obtained from BOINC and XWHEP simulators we developed. We have considered the public volunteer computing project SETI@Home (`seti`) ran by BOINC[20], and three private Desktop Grid deployments at University Notre Dame, ran by Condor[21] (`nd`), and University Paris XI, ran by XtremWeb[22] (`upsud`) as simulated DGs. The availability traces of those DGs are provided by the Failure Trace Archive[23]. Many other works have been proposed in the literature, which study host volatility [24, 25, 26, 27].

To emerge a common behavior, all the execution profiles, corresponding to 4000 BoT simulations are aggregated in Figure 8. Because of hosts volatility and the high variation in the number of computing host, it is expected that DGs have great variation in their execution time compared with an infrastructure which would offer constant computing capabilities. The slowdown metric compares the actual BoT task completion time at the end of the BoT execution versus an estimated completion time. The estimated completion time is calculated by multiplying by two the elapsed time at 50% of the BoT execution (hence assuming a constant completion rate). One can observe that the distribution is largely skewed and the completion time slowdown often exceeds 100%. This is due to host volatility and the fact that Desktop Grid middleware have to wait for failure detection before reassigning tasks. We consider the tail as the part of the BoT executed

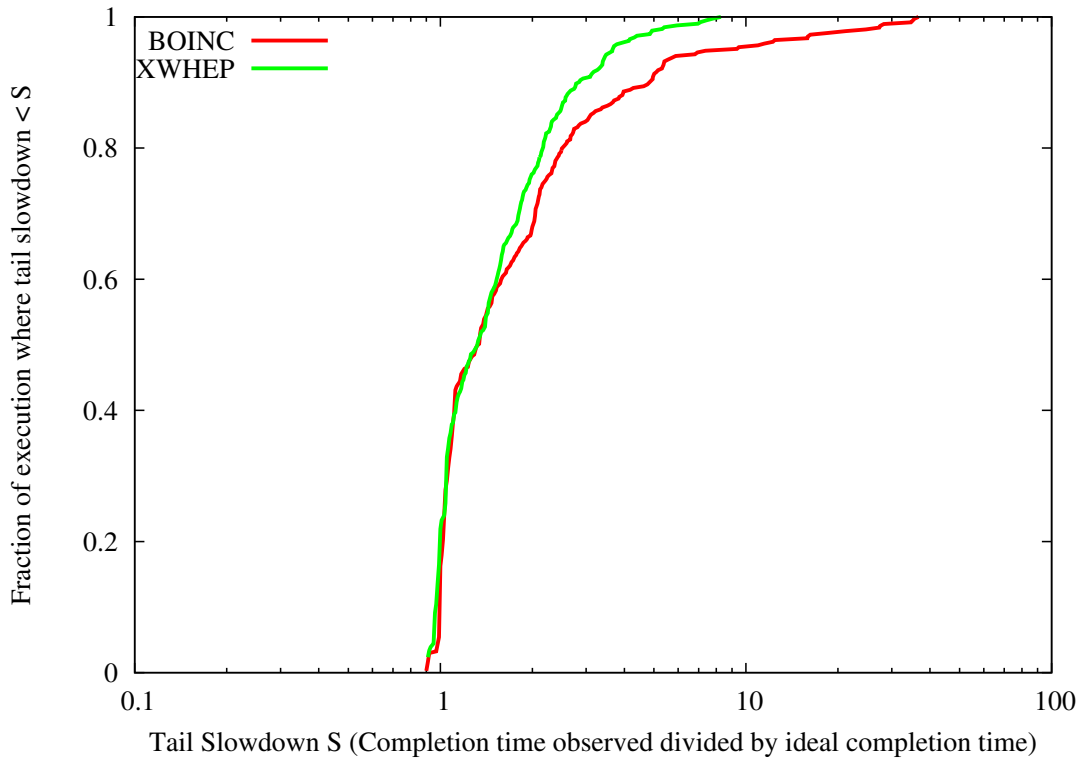


Figure 8: Profiling execution of BoTs in DG: slowdown is the BoT completion time divided by the estimated completion time at 50% of the BoT execution. The x-axis presents the slowdown in log scale, the y-axis presents the CDF of BoT slowdown.

after an elapsed time of 150% of the estimated execution time (corresponding to a slowdown of 150% and more), we can see that, depending on the availability trace and middleware considered, between 20% and nearly all of BoT executions were affected by a tail. Among those BoT, table 3 shows the fraction of tasks executed during the tail period.

Table 2: Percentage of jobs in the tail

	BOINC		XtremWeb	
	mean (%)	std	mean (%)	std
`seti`	8	0.4	4.5	2.2
`nd`	4.7	1.3	3.1	1.2
`upsud`	2.7	1.1	0	0

This analysis emphasizes the negative impact of the tail effect on DG users QoS. A large number of BoTs are affected by a tail effect that slows the completion time significantly, but only a small part of the BoT jobs are concerned by this tail.

10 Large Deployment Using Grid5000 and Amazon EC2 as an Experimental Platform

To test SpeQuloS and evaluate its performance under realistic conditions, we performed experimentations using Grid5000 and Amazon EC2. In DGs, from few hundred to several thousand worker nodes are participating to computational effort during several days. To reproduce this situation, we used Grid5000 resources as DG workers. In this section, we are presenting this experimentation and its results.

10.1 SpeQuloS and the XtremG5K project

Our experimentation is supported by the XtremG5K project, which main goal is to use Grid5000 to provide computational resources to an XWHEP DG. The XtremG5K project is composed of several components:

- An XWHEP server, hosted at Laboratoire de Recherche en Informatique (LRI). It can accept jobs submission from EGEE users through the 3G Bridge [3] 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 LRI XWHEP server.
- A set of XWHEP workers, executed on Grid5000 nodes. In the experimentation presented in this report, a Grid5000 job is considered as a pilot job which runs one or several XWHEP workers. To approximate a voluntary-based and best effort participation to computation as observed with DG worker nodes, we decided to use the "best-effort" queue of Grid5000.

In this experimentation, SpeQuloS was connected to the LRI XWHEP server, and to the Amazon EC2 Cloud service. It does not need to communicate with Grid5000. The diagram 9a shows the experimentation architecture.

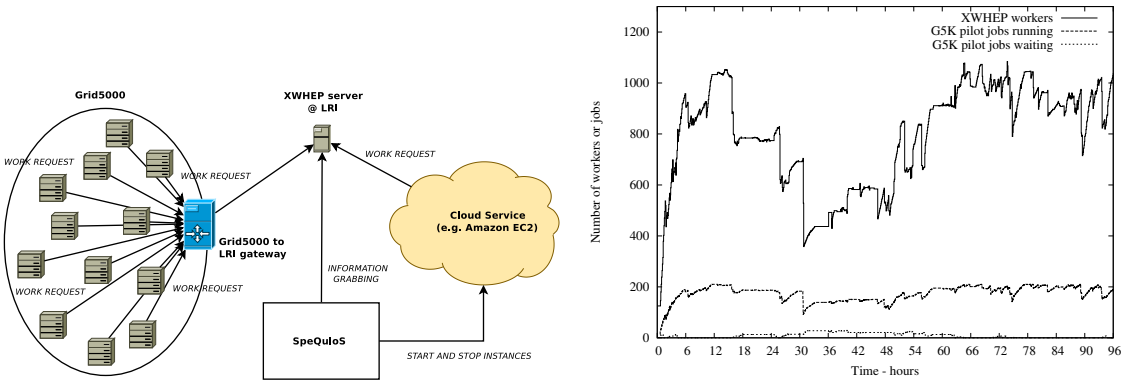
10.2 Grid5000 resources usage

The experimentation has been deployed on seven Grid5000 sites (Lyon, Grenoble, Bordeaux, Lille, Nancy, Sophia and Toulouse). On each site, we ran the algorithm 3 for pilot job submission. Each pilot job, denoted as "start_XWHEP_workers" in the algorithm, is submitted to one Grid5000 node and starts one XWHEP worker per CPU on the node executing it.

We ran this experimentation during 4 days. The figure 9b 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 lead 400 to more than 1000 workers to participate to the DG. The number of Grid5000 pilot jobs waiting for submission on all sites simultaneously never exceeds 30.

10.3 Experimentation results

We ran several experimentation scenarios with or without using SpeQuloS. In each scenario, an XWHEP user submits a BoT containing from 1000 to 10000 jobs. Each individual job needs few

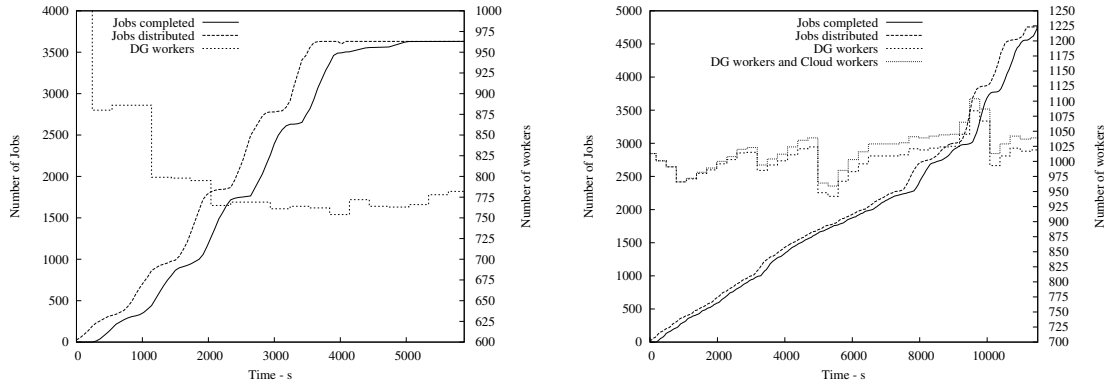


(a) The XtremG5K project and SpeQuloS architecture (b) Number of XWHEP workers and Grid5000 pilot jobs running and waiting during the experimentation

Figure 9: 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
```



(a) SpeQuloS not used

(b) SpeQuloS used

Figure 10: Submission and completion of jobs with number of desktop grid workers, according to the scenario time

minutes to be computed by a worker. Jobs are distributed by the LRI XWHEP server to worker nodes hosted in Grid5000 and to Cloud workers from Amazon EC2 when SpeQuloS was used. In this section, we are presenting the results of two of these scenarios.

Figure 10a 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 distributed and completed as well as the number of DG workers, according to the scenario time. The number of jobs distributed denotes the number of jobs that have been sent to a DG worker. 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, caused by jobs submitted to DG workers who then leave the DG. In that case, the DG server has to resubmit the job to another worker, but the time required for this operation can be significant, as the missing workers have to be detected first.

By provisioning stable resources from Cloud, one goal of SpeQuloS is to address this problem. Figure 10b shows similar results from a scenario where SpeQuloS was 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 some Cloud workers to participate to the DG. 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.

These experimentations are not sufficient to draw any general conclusion about the ability of SpeQuloS to mitigate the tail effect. A lot of parameters still have to be investigated, and some reproducible set of experiments have to be conducted. However, the experimentations presented in this section demonstrate the ability for SpeQuloS to monitor remotely a Desktop Grid and to

Table 3: State of the Grid5000 libcloud driver implementation. Implemented features are marked Ok and unimplemented features are marked WIP (Work In Progress)

Provider	**Create Instance**	**Reboot Instance**	**Destroy Instance**	**List Disk Images**	**Deploy Disk Images**	**List Node Size**
Grid5000	Ok	Ok	Ok	WIP	WIP	WIP

deploy remotely Cloud workers on Grid5000, without any measurable overhead.

10.4 Using Grid5000 as a private Cloud thanks to SpeQuloS

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 cost of public Cloud services such as Amazon EC2 and 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 features of common Cloud services.

10.5 The Grid5000 libcloud driver

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[28] to implement a new libcloud <<driver>> to access to Grid5000 resources. This implementation allows SpeQuloS to interact with Grid5000 as a typical IaaS Cloud service, with the same API as any other Cloud service for which driver exists in libcloud. The current state of this implementation is shown is table 3. It is currently possible to interact with a Grid5000 Cloud instances (Create Instance, Reboot Instance and Destroy Instance features) but only using the default Grid5000 disk image (List Disk Image and Deploy Disk Image features) without dealing with nodes specifications, such as number of CPU (List Node Size feature). The implementation of those missing features is underway.

With the Grid5000 libcloud driver, a large amount of Cloud resources is made available to researchers. Software which use the libcloud API can benefit of Grid5000 resources without any additional development. It is also possible to combine Grid5000 and other Cloud services in the same experiment. Future work will extend this service to the 3G bridge, so that the whole EDGI infrastructure could be used similarly.

10.6 Experimentation results

To validate the SpeQuloS multi-Cloud feature, 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 section 10.

Figure 11 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.

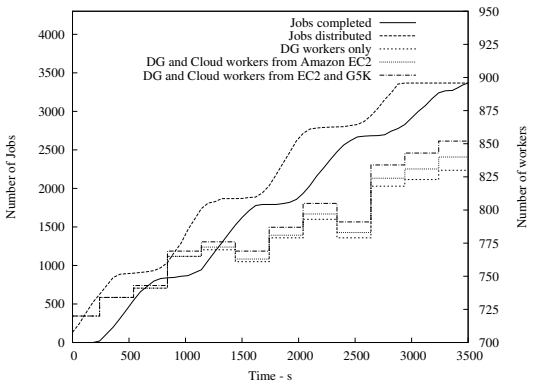


Figure 11: Submission and completion of jobs with number of desktop grid workers, according to the scenario time. Grid5000 resources are used as Cloud workers

10.7 Using BOINC and SpeQuloS to execute computations on Amazon EC2 Spot Instance

In the Cloud, Amazon has recently introduced EC2 Spot instances[29] where users can bid for unused Amazon EC2 instances. Usually, the price for Spot Instance is less than the corresponding regular instance price. If the market spot price goes under the user's bid, user gains access to available instances. In this section, we show an experiment where BOINC workers are hosted on Amazon Spot instances. SpeQuloS monitors the execution on the Spot instances, and if it detects that too many Spot instances terminate, it provisions regular instances to finish the computation. As in DG, spot resources are variable and unreliable. As the price market fluctuates, the number of workers varies accordingly. Workers leave the system without notice, similarly to Desktop PCs. Tail effect is also present with Amazon EC2 spot instances. The aim of this experiment is to show the SpeQuloS ability to bring QoS to unreliable infrastructure based on the BOINC middleware.

Amazon EC2 spot instances are only started if a user bid is higher than a variable price. Thus, with Amazon EC2 spot instance, the host availability depends both on the user's bids and the instance price market variation. We consider the following usage of EC2 Spot instance: a total renting cost per hour (C) is set by the user to use several instances. Because the total cost is constant while the price market fluctuates, the number of instances provisioned varies. We adopt the following bid strategy to implement this usage scenario: We place a sequence of n bids at price $\frac{C}{i}$, where $i \in 1..n$. n should be chosen so that $\frac{C}{n}$ 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 C. 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, we have generated the instances availability traces of the *c1.medium* instance for a renting cost of 10 dollars per hour.

We deployed an experimentation implementing this scenario for a period of 7 days. The BoT was composed of 10000 tasks of the DSP application. The BOINC server was deployed on a regular instance and each of the spot instance ran a BOINC client that connects to this server. We used the biding strategy already presented in section 2.1 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.

SpeQuloS modules were deployed on the same host than the BOINC server. We configured

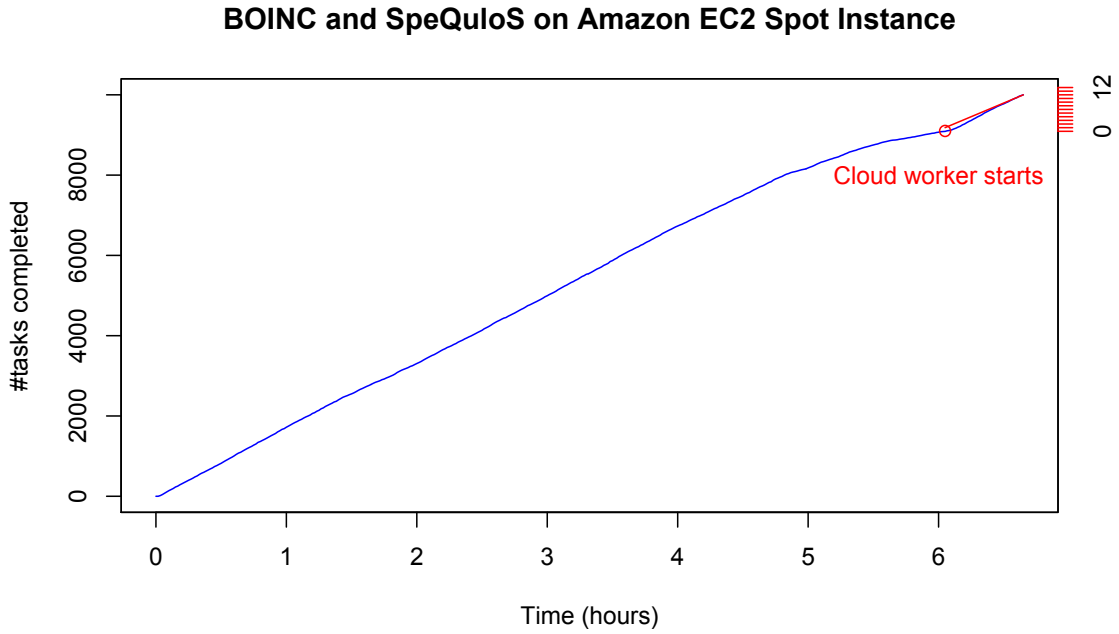


Figure 12: Execution of a BoT using Amazon EC2 spot instances and SpeQuloS

SpeQuloS to start some Amazon EC2 regular instances as Cloud worker to support the BoT execution.

Figure 12 shows an example of BoT execution on our platform. We can see that SpeQuloS starts some Cloud workers at the 90th percent of completion of the BoT. The BoT execution took 6.65 hours to be completed.

This experiment denotes the ability for SpeQuloS to mitigate the tail effect and bring some QoS support to an unreliable environment executing the BOINC middleware.

11 Conclusions

JRA2 has successfully developed a first version of the SpeQulos prototype to improve QoS for BoT application in the EDGI infrastructures. The four components which compose the SpeQulos middleware are the QoS Info, the QoS Credit System, the Oracle and the QoS Scheduler.

SpeQuloS allows:

- to provide real-time Desktop Grid information on BoT execution and archive BoT execution traces
- to account for Cloud usage and to implement Cloud resources arbitration policies
- users to monitor their BoT and to obtain predicted execution time, with and without the tail effect.
- users to specifies that QoS is required for some of their BoT and allocate virtual credits to implement QoS

- to provision additional Cloud resources dedicated to the execution of a specific BoT

Although, statistically we don't have enough results to draw definitive conclusion about SpeQuloS efficiency with respect to QoS management, we think that early large scale experiments have demonstrated the features and stability of the prototype and show basic QoS performance improvement.

11.1 Measures/indicators

One indicator to assess the QoS effort for this deliverable is the number of Cloud supported and the objective is set to one, as we currently support Eucalyptus clouds used in EDGI infrastructure. We have made extensive tests using Amazon EC2, which is compatible with Eucalyptus. In addition we provided support for Rackspace and Grid5K. Although Grid5K is not exactly defined a Cloud, we have shown that SpeQulos can be extended to support several Cloud technologies.

The other indicator for this deliverable is the number of Desktop Grid technologies supported and the objective one technology is reached, as we currently support both BOINC and XtremWeb.

11.2 Publications

The project has published papers [30, 31, 32, 33, 34] in the following venues:

- *Workers in the Clouds* Attila Csaba Marosi and Peter Kacsuk PDP 2011 - The 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, Ayia Napa, Cyprus
- *Hybrid Distributed Computing Infrastructure Experiments in Grid5000: Supporting QoS in Desktop Grids with Cloud Resources* – S. Delamare, G. Fedak, D. Kondo and O. Lodygensky . – Grid500 Spring School (peer reviewed), 2011. **[Best presentation award]**
- *Towards Hybrid Desktop Grid and Cloud Infrastructures* – G. Fedak and S. Delamare – Book Chapter – Desktop Grid Computing – CRC Press, 2011.
- *SpeQuloS: A QoS Service for BoT Applications Using Best Effort Distributed Computing Infrastructures* – S. Delamare, G. Fedak, D. Kondo and O. Lodygensky . in International Symposium on High Performance Distributed Computing (HPDC'2012), Delft, Nederlands, 2012
- *SpeQuloS: A QoS Service for Hybrid and Elastic Computing Infrastructures.* S. Delamare, G. Fedak, D. Kondo, and O. Lodygensky. Journal of Cluster Computing (Cluster), SI: Selected paper from HPDC'12, 2013

11.3 Limitations and future tasks

Because of the strict time constraints and the short release cycle, our prototype has the following main limitations, which will be addressed in the second year of the project:

- SpeQuloS integration with other components of EDGI needs to be strengthened. In the future, we will implement the QoS Info using the probes developed by JRA1. Also, we will use the ability of the 3G bridge to launch Cloud resources developed by JRA1 during the first period of the project.
- Because SpeQuloS is not running yet in the production infrastructure, we are lacking measurement to evaluate our strategies. In the second period of the project, we will use traces and simulations to propose new Oracle heuristics.

11.4 Acknowledgment

SpeQuloS and EDGI (European Desktop Grid Initiative) receives Community funding from the European Commission within Research Infrastructures initiative of FP7 (grant agreement Number RI-261556).

Experiments presented in this deliverable were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

We thank l'Institut des Grilles for having supported our project XtremG5K through the interface projects between production grid and experimental grid.

A SpeQuloS resources

All the resources are available in the EDGI project SVN.

A.1 SpeQuloS package

SpeQuloS 0.2.1 is available since 3rd March 2011. It can be downloaded at:
`http://graal.ens-lyon.fr/~sdelamar/spequlos/spequlos-0.2.1.tar.gz`

A.2 SpeQuloS installation documentation

The installation documentation is provided in the SpeQuloS package. It can be found in the file "install.txt", under the "docs" subdirectory of the archive. It can also be found at the url: `http://graal.ens-lyon.fr/~sdelamar/spequlos/docs/install.txt`

A.3 SpeQuloS usage documentation

The user documentation is provided in the SpeQuloS package. It can be found in the file "documentation.pdf", under the "docs" subdirectory of the archive. It can also be found at the url: `http://graal.ens-lyon.fr/~sdelamar/spequlos/docs/documentation.pdf`

A.4 Cloud Worker image creation documentation

The documentation describing how to build a Cloud Worker image to be used by a Cloud service is provided in the SpeQuloS package. It can be found in the file "how_to_create_a_cw_vm.txt", under the "docs" subdirectory of the archive. It can also be found at the url: `http://graal.ens-lyon.fr/~sdelamar/spequlos/docs/how_to_create_a_cw_vm.txt`

A.5 BOINC server patching documentation

The documentation describing how to patch a BOINC server to be supported by SpeQuloS is provided in the SpeQuloS package. It can be found in the file "install_boinc_for_spequlos.txt", under the "docs" subdirectory of the archive. It can also be found at the url: `http://graal.ens-lyon.fr/~sdelamar/spequlos/docs/install_boinc_for_spequlos.txt`

A.6 SpeQuloS step by step example

A step by step example of a typical Spequlos use case is presented in a document available at the url: `http://graal.ens-lyon.fr/~sdelamar/spequlos/docs/step_by_step.pdf`

References

[1] E. Urbah, P. Kacsuk, Z. Farkas, G. Fedak, G. Kecskemeti, O. Lodygensky, A. Marosi, Z. Balaton, G. Caillat, G. Gombas, A. Kornafeld, J. Kovacs, H. He, and R. Lovas. EDGeS: Bridging EGEE to BOINC and XtremWeb. *Journal of Grid Computing*, 7(3):335–354, September 2009.

[2] Zoltan Balaton, Zoltan Farkas, Gabor Gombas, Peter Kacsuk, Robert Lovas, Attila Csaba Marosi, Ad Emmen, Gabor Terstyanszky, Tamas Kiss, Ian Kelley, Ian Taylor, Oleg Lodygensky, Miguel Cardenas-Montes, Gilles Fedak, and Filipe Araujo. EDGeS: the Common Boundary Between Service and Desktop Grids. *Parallel Processing Letters*, 18(3):433–453, September 2008.

[3] Haiwu He, Gilles Fedak, Peter Kacsuk, Zoltan Farkas, Zoltan Balaton, Oleg Lodygensky, Etienne Urbah, Gabriel Caillat, and Filipe Araujo. Extending the EGEE Grid with XtremWeb-HEP Desktop Grids. In *Proceedings of CCGRID'10, 4th Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid 2010)*, pages 685–690, Melbourne, Australia, May 2010.

[4] Attila Csaba Marosi, Peter Kacsuk, Gilles Fedak, and Oleg Lodygensky. Sandboxing for Desktop Grids Using Virtualization. In *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing PDP 2010*, pages 559–566, Pisa, Italy, February 2010.

[5] Gabriel Caillat, Oleg Lodygensky, Gilles Fedak, Haiwu He, Zoltan Balaton, Zoltan Farkas, Gabor Gombas, Peter Kacsuk, Robert Lovas, Attila Csaba Maros, Ian Kelley, Ian Taylor, Gabor Terstyanszky, Tamas Kiss, Miguel Cardenas-Montes, Ad Emmen, and Filipe Araujo. EDGeS: The art of bridging EGEE to BOINC and XtremWeb. In *Proceedings of Computing in High Energy and Nuclear Physics (CHEP'09) (Abstract)*, Prague, Czech Republic, March 2009.

[6] P. Kacsuk, Z. Farkas, and G. Fedak. Towards Making BOINC and EGEE Interoperable. In *Proceedings of 4th IEEE International Conference on e-Science (e-Science 2008), International Grid Interoperability and Interoperation Workshop 2008 (IGIIW 2008)*, pages 478–484, Indianapolis, USA, December 2008.

[7] Gabriel Caillat, Gilles Fedak, Haiwu He, Oleg Lodygensky, and Etienne Urbah. Towards a Security Model to Bridge Internet Desktop Grids and Service Grids. In *Proceedings of the Euro-Par 2008 Workshops (LNCS), Workshop on Secure, Trusted, Manageable and Controllable Grid Services (SGS'08)*, Las Palmas de Gran Canaria, Spain, August 2008. 247–259.

[8] Edgi: European desktop grid initiative project. `http://edgi-project.eu`, 2011.

[9] David Anderson. BOINC: A System for Public-Resource Computing and Storage. In *proceedings of the 5th IEEE/ACM International GRID Workshop*, Pittsburgh, USA, 2004.

[10] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky. Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid. *Future Generation Computer Systems*, 21(3):417–437, mar 2005.

[11] David Anderson and Gilles Fedak. The Computational and Storage Potential of Volunteer Computing. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 73–80, Singapore, May 2006.

[12] Oleg Lodygensky, Gilles Fedak, Franck Cappello, Vincent Neri, Miron Livny, and Douglas Thain. XtremWeb & Condor : Sharing Resources Between Internet Connected Condor Pools. In *Proceedings of CCGRID'2003, Third International Workshop on Global and Peer-to-Peer Computing (GP2PC'03)*, pages 382–389, Tokyo, Japan, 2003. IEEE/ACM.

[13] Oleg Lodygensky, Gilles Fedak, Vincent Neri, Cécile Germain, Franck Cappello, and Alain Cordier. Augernome & XtremWeb : Monte Carlo Computation on a Global Computing Platform. In *Proceedings of CHEP03 Conference for Computing in High Energy and Nuclear Physics*, San Diego, USA, 2003.

[14] Gilles Fedak, Cécile Germain, Vincent Néri, and Franck Cappello. XtremWeb: A Generic Global Computing Platform. In *Proceedings of 1st IEEE International Symposium on Cluster Computing and the Grid CCGRID'2001, Special Session Global Computing on Personal Devices*, pages 582–587, Brisbane, Australia, May 2001. IEEE/ACM, IEEE Press.

[15] Fabrizio Gagliardi, Bob Jones, François Grey, Marc-Elian Bégin, and Matti Heikkurinen. Building an infrastructure for scientific grid computing: Status and goals of the egee project. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 363(1833):1729–1742, 2005.

[16] Mark Silberstein, Artyom Sharov, Dan Geiger, and Assaf Schuster. Gridbot: Execution of bags of tasks in multiple grids. In *SC '09: Proceedings of the 2009 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2009. ACM.

[17] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010)*, Melbourne, Australia., May 2010.

[18] libcloud, a unified interface to the cloud. `http://incubator.apache.org/libcloud/`, 2011.

[19] Derrick Kondo, Bahman Javadi, Paul Malecot, Franck Cappello, and David P. Anderson. Cost-benefit analysis of cloud computing versus desktop grids. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–12, Washington, DC, USA, 2009. IEEE Computer Society.

[20] B. Javadi, D. Kondo, JM. Vincent, and D.P. Anderson. 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)*, September 2009.

[21] Brent Rood and Michael J. Lewis. Multi-state grid resource availability characterization. In *8th Grid Computing Conference*, September 2007.

[22] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Comp. Syst.*, 23(7):888–903, 2007.

[23] Derrick Kondo, Bahman Javadi, Alex Iosup, and Dick Epema. 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.

[24] Derrick Kondo, Felipe Araujo, Paul Malecot, Patricio Domingues, Luis M. Silva, Gilles Fedak, and Franck Cappello. Characterizing Result Errors in Internet Desktop Grids. In *European Conference on Parallel and Distributed Computing EuroPar'07*, Rennes, France, August 2007.

[25] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. Resource Availability in Enterprise Desktop Grids. *Future Generation Computer Systems*, 23(7):888–903, August 2007.

[26] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. On Resource Volatility in Enterprise Desktop Grids. In *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (eScience'06)*, pages 78–86, Amsterdam, Netherlands, December 2006.

[27] Paul Malécot, Derrick Kondo, and Gilles Fedak. XtremLab: A System for Characterizing Internet Desktop Grids. In *Poster in The 15th IEEE International Symposium on High Performance Distributed Computing HPDC'06*, Paris, France, June 2006.

[28] Grid'5000 API. `https://api.grid5000.fr/`, 2011.

[29] Amazon Web Services. An introduction to spot instances. Technical Report API version 2009-11-30, Amazon Elastic Compute Cloud, 2009.

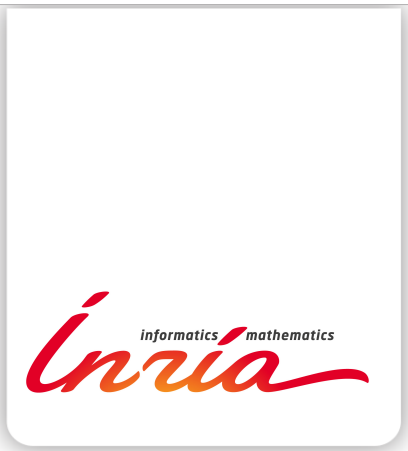
[30] Simon Delamare, Gilles Fedak, Derrick Kondo, and Oleg Lodygensky. SpeQuloS : A QoS Service for Hybrid and Elastic Computing Infrastructures. *Journal of Cluster Computing*, 2013.

[31] Simon Delamare, Gilles Fedak, Derrick Kondo, and Oleg Lodygensky. 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)*, pages 173–186, Delft, The Netherlands, June 2012.

[32] Simon Delamare and Gilles Fedak. SpeQulos: A Framework for QoS in Unreliable Distributed Computing Infrastructures using Cloud Resources. In *Grid5000 Spring School*, Reims, France, February 2011. **Best Presentation Award**.

[33] Simon Delamare and Gilles Fedak. Towards Hybridized Clouds and Desktop Grid Infrastructures. In Christophe Cérin and Gilles Fedak, editors, *Desktop Grid Computing*, pages 261–285. Chapman & All/CRC Press, 2012.

[34] Oleg Lodygensky, Etienne Urbah, Simon Dadoun, Anthony Simonet, Gilles Fedak, Simon Delamare, Derrick Kondo, Laurent Duflot, and Xavier Garrido. FlyingGrid : from Volunteer Computing to Volunteer Cloud. In *poster in Computing in Hign Energy and Nuclear Physics (CHEP'12)*, New York, USA, 2012.



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