Soft Comput manuscript No.
(will be inserted by the editor)

A simple model to exploit reliable algorithms in cloud federations

A. J. Rubio-Montero · M. A. Rodríguez-Pascual · R. Mayo-García

Received: date / Accepted: date

Abstract Exploiting resources belonging to multiple cloud providers in an efficient way is still an open issue for distributed computing. Scheduling algorithms based on heuristic, probabilistic, queue theory, or complex soft computing methods are suitable to tackle the heterogeneity and dynamism present in cloud federations. Nevertheless, the available brokering tools are focused on the deployment of services on-demand. The systems able to accomplish high throughput calculations, such as the pilot-job systems, do not support the inclusion of these algorithms due to their lack of adaptability. The recently implementation of cloud drivers for the GWpilot framework allows developers to profit from its flexibility, compatibility and scheduling features. Moreover, the framework allows the personalised characterisation of cloud resources that those algorithms require, overcoming their lack of trustworthiness in the information provided by the cloud services. In this work, a simple model together with a methodology to couple scheduling software with GWpilot is presented. To demonstrate the suitability of the approach, a legacy self-scheduler specialised on reliable executions in dynamic environments has been stacked and tested on the EGI FedCloud infrastructure with the Nagano legacy application.

Keywords Distributed Computing · pilot jobs · modelling · IaaS

1 Introduction

IaaS cloud providers are heterogeneous and their real availability dynamically changes through the time (Foster et al, 2008). Thus, the allocation of virtual machines (VMs) across multiple providers is a NP-complete problem(Garey and Johnson, 1979) that can be tackled with the sub-optimal approaches that soft computing or the operative research (Pinedo, 2005) can provide. For example, meta-heuristics (Gómez-Iglesias et al, 2010), evolutionary algorithms (Nesmachnow et al, 2010), or fuzzy logic (Saleh, 2013) have demonstrated their suitability for

A. J. Rubio-Montero, M. A. Rodríguez-Pascual, R. Mayo-García .
CIEMAT. Av. Complutense 40, 28040. Madrid, Spain.
E-mail: antonio.rubio@ciemat.es

improving calculations on large distributed environments such as grids. Moreover, cloud federations are more dynamic and complex than grids because more parameters are taken into account for scheduling(Aceto et al, 2013; Sheikhalishahi et al, 2015), although on the other side, the benefits that virtualisation provides can be higher. In this sense, some brokers, such as one proposed by Anastasi et al (2014), take advantage of specific monitoring to reduce the complexity of VM deployments to improve the quality of service through the time. Nevertheless, current cloud brokering mechanisms are focused on obtaining as many computational resources as possible with a limited (computational and/or economic) cost (Tordsson et al, 2012; Yangui et al, 2014) and with no support to legacy high throughput computing (HTC) applications and to those kinds of algorithms. Moreover, the information systems (IS) in cloud federations do not provide an accurate description of the status of cloud providers as happened with grids. Furthermore, as in grid federations, the resources fail. An analysis on the source of failures on cloud environments and the most common ways of addressing them is available in (Zhani and Boutaba, 2015). Besides, it is clear that a developer of any adaptive algorithm should take account of the localisation and performance of the resources to reduce the final makespan of the calculations.

In this sense, the makespan of any distributed application depends on the turnaround time of every task. Turnaround is specially influenced by overheads when short-duration tasks are scheduled. In contrast, long-duration tasks suffer from more failures that waste computational time. To calculate the turnaround, developers should estimate the time required by stage-in and -out operations, the performance offered, the failures, and the overheads related of providers and middleware. For this purpose, it is needed a correct characterisation of the providers, which cannot be possible without the appropriation and monitoring that pilot-job technique could offer. Nevertheless, current pilot systems neither support the customisation of monitoring nor report their own generated overheads.

This drawback is overcome with GWpilot. This framework now supports cloud resources (Rubio-Montero et al, 2015c), preserving all its demonstrated features (Rubio-Montero et al, 2015b); among them, it allows monitoring personalised aspects of the computation and it performs a true provisioning among multiple cloud providers on-demand. Furthermore, the capacity of completely characterize its added overhead is a key feature to formulate a trustworthy model for task turnaround. This model, together with the GWpilot support for task scheduling, allows stacking specialised scheduling tools to the framework, which can properly now run on both grid and cloud infrastructures.

In this sense, dynamic self-schedulers are the most feasible tools for this purpose because they are able to adapt some specific calculation according to the estimation of the infrastructure status, but following their own criteria and disregarding the information from the IS when needed. In this work, a self-scheduling framework (Rodríguez-Pascual et al, 2013) devoted to improve the reliability of Monte Carlo (MC) codes has been stacked. Although the framework has previously demonstrated its standalone performance on grids (Rubio-Montero et al, 2015d), it will take advantage of the proper characterisation of clouds offered by GWpilot. The self-scheduling algorithm requires knowing the theoretical performance of the infrastructure for computing a concrete application before assigning a number of samples to be computed by a certain resource. This performance is calculated following a mathematical model based on turnaround, which is continuously fitted with the benchmarking information retrieved from ending tasks. Therefore, this example can be used as a guide for the future stacking of algorithms based on turnaround and reliability as the ones mentioned in Section 2.3.

Summarizing, a simplified task turnaround model that takes into account the underlying overheads is presented in this work. Additionally, a methodology to incorporate external scheduling into GWpilot is presented. The suitability of the presented approach is demonstrated through the statistical analysis of the performed executions of a real MC application on the in-production FedCloud infrastructure, comparing the results of the stacked self-scheduler to the ones achieved without using it.

2 The persistent characterisation issue

2.1 Information systems in federations

Client systems of cloud infrastructures highly rely on the information provided by geographically distributed sites. By doing so, users, developers and administrators are aware of when, where and how to request virtual machines. Such information is shown by the information systems, locally placed as one of the several components that form a cloud site, the data of which should be compiled at top-level for general queries. Thus, the standardisation of these services are one of the foundations of a cloud federation (Foster et al, 2008; Moreno-Vozmediano et al, 2012) as opposite to multi-cloud approaches (Grozev and Buyya, 2014), focused on the interoperation through different protocols. In principle, any system or user wanting to launch VMs on a particular cloud federation will distribute the requests according to the information that the main IS compiles regarding the available resources. Unfortunately, sometimes such information is not centralised in any way, and even it is usually not accurate. The standardised interfaces for monitoring are based on OCCI and on LDAP, following the GLUE schema (Andreozzi et al, 2009). Currently, the former is only deployed as a local service at the cloud provider. The latter is both local as well as a shared service in federations such as FedCloud[1]. However, the characterisation of resources performed by these services is incomplete and defective. It prevents properly estimating task turnaround and consequently, a resilient and efficient VM deployment among cloud providers in a federation.

2.2 Characterisation for scheduling

Task turnaround is the basic model that supports any of the scheduling algorithms suitable for distributing calculations. In this sense, a widely accepted and simplified definition of turnaround (Mościcki, 2011; Montero et al, 2006) follows the equation:

$$T = T_{sched} + T_{xfer} + T_{exec} \tag{1}$$

where:

[1] https://www.egi.eu/infrastructure/cloud/

- T_{exec} is the effective execution time of the application in the remote resource. It depends on the hardware, software, overload and reliability offered by the cloud provider for the concrete application, as well as on the BoT size.
- T_{xfer} stands for the time wasted in transferring inputs, outputs and software. It depends on the protocols used and the bandwidth of the remote cloud provider, as well as on the amount of data to transfer.
- T_{sched} comprises the time wasted in Job Scheduling processes. It depends on the infrastructure overload, the allowed utilisation (quotas) and the middleware implementation.

Consequently, the suitability and preference for any cloud provider depends on its *real availability* for a type of calculation at certain time, which is compound by:

(a) the volume of the hardware resources and the software actually offered: the number of cores, memory, storage, bandwidth, data, libraries, licenses, etc., as well as the VM images available, i.e. the *resource_tpl* and the *os_tpl* templates (according to the OCCI specification) and the instances allowed;
(b) the effective performance of these resources for the application, i.e. the profile of the application on that hardware, as well as the potential associated costs and the granted percentage utilisation (as in virtualized systems hardware is usually shared).
(c) the reliability of the application, the middleware, and the provider itself (that depends on the configuration, maintenance, security, network supplier, etc.).

Nevertheless, as the date of publishing this work, the information belonging to point (a) is only partially available at the IS deployed in federations. The rest of points are practically discarded. In particular, the GLUE scheme implemented in top-BDIIs currently does not allow knowing (among others aspects):

1. the capacity and filling rate of a cloud provider;
2. the quotas established in cloud providers for a user belonging to certain VO;
3. the exact hardware or reliable benchmarks of every virtualisation hosts;
4. statistics about the QoS or the SLA achievement for every provider.

It is noteworthy to mention that GLUE is extensible and currently counts on tags that declare some of the aforementioned characteristics. Nevertheless, the information provided is usually incomplete, not updated and erroneous in many cases. Additionally, way more aspects should be taken into account in cloud environments (Aceto et al, 2013) than in grids. Nevertheless, the grid experience with GLUE shows that many characterisation aspects will never be implemented. Obviously, a scheduling system working with cloud providers can follow a multi-cloud approach (Grozev and Buyya, 2014), monitoring every provider through its OCCI interface. Although extensions of the specification have been proposed (Mohamed et al, 2015), the OCCI v1.1 specification does not even provide the aforementioned information.

Moreover, the accounting and monitoring tools deployed in cloud federations do not aggregate their statistics to the IS, and also cannot be directly interfaced because only visual tools are available (Zanikolas and Sakellariou, 2005). In this sense, standardisation advances are carrying on (Ciuffoletti, 2014), but there are not implementations to evaluate their suitability.

Therefore, the deployment of any adaptive scheduling algorithm is hindered by this lack of characterisation required to estimate the task turnaround. Considering the aforementioned issues, the possibilities are really limited for the systems following early-binding approaches. In contrast, late-binding techniques can solve many of the commented characterisation problems and they can provide more advantages, as will be explained in Subsection 2.4.

2.3 Related approaches

Modelling cloud infrastructures according to an optimum turnaround represents a clear step forward towards obtaining an efficient calculation. However, dynamism of resources, specially unexpected failures and slowdowns, implies rely on adaptive techniques to reach a resilient calculation. Although they can be combined, both orientations are usually differenced by authors:

Scheduling based on turnaround. Despite the lack of characterisation tools, some work has been done on this topic. For example, techniques have been implemented to obtain a better task allocation (Panda et al, 2015) based on turnaround in multi-cloud environments. The Divisible Load Theory has been used(Abdullah and Othman, 2013) as well as other methodologies such as classical replication of tasks (Sajid and Razaa, 2015), workflow scheduling (Smanchat and Viriyapant, 2015) or even simple AVL-trees (Chiu et al, 2014). Making the most of soft computing, particle swarm optimization (Xu et al, 2015), bee colonies (Babu and Venkata, 2013) and genetic algorithms (Wang et al, 2016; Tao et al, 2014) are used, but all those focus on the placement of VMs in single IaaS providers. Unlike last approaches, game theory (Shie et al, 2014) can also be applied to federations.

Scheduling based on autonomic computing. The first aspect to consider is to improve the fault tolerance of the execution processes. This can be done by coping failure-prone environments with a multi-hybrid (Moon and Youn, 2015) and autonomic job scheduling (Bala and Chana, 2015). Robustness is provided by task replication based on heuristics for the first approach, and by migrating the VM automatically in case of task failure occurrences due to the over-utilization of resources in the last one. Additionally, taking into account failures (Zhani and Boutaba, 2015) is useful for the autonomous reconfiguration of sites (Mohamed et al, 2015). Moreover, monitoring performance benchmarks, SLA and failures can be combined (Lu et al, 2016). A step forward can be performed if QoS, VM migrations and budgets are balanced (Lucas-Simarro et al, 2015) to autonomously deploy a virtualised HTC cluster among commercial providers. Related to the work presented in this paper, a model to evaluate the reliability of cloud infrastructures and maximize the resource usage and its behaviour executing a non-sequential MC simulation can be found in (Snyder et al, 2015), but it does not consider the performance requirements of particular applications.

As a closing remark, it is important to notice that most of these mentioned solutions are only tested or analysed in simulators, or make use of single cloud providers (such as AWS[2]) without performing a distribution of tasks among several ones, or even they do not interface with the standardised services required in federations, making difficult to determine its usefulness on real environments. Moreover, most of autonomic scheduling does not take account of turnaround. It is very difficult for developers to improve by their own the execution of concrete HTC applications if a completely autonomous system is used. Tasks are not consolidated services and they cannot estimate a priori how long they will last (or how many they will expend) in a certain provider because the offered hardware is different.

Unlike previous solutions, the approach described in this work is not only designed to maximise the VM usage and deployment among cloud providers, but to support the scheduling algorithms that developers implement for their applications. This is possible by decoupling the turnaround model from the IaaS provisioning, but supporting a guided provisioning and standardised interfaces. Thus, the performance on real time can be estimated for a concrete application and used by third-party schedulers to improve the reliability and reduce the final makespan of the calculation, as will be demonstrated through next sections.

2.4 Resource Provisioning with pilot jobs

Pilot systems have been extensively used through years to increase the reliability and throughput in grid environments. On the other hand, the provisioning mechanisms currently available in IaaS clouds are focused on setting up VMs for the consolidation of services. However, they do not prevent running pilot jobs as temporal services (Mhashilkar et al, 2014; Luckow et al, 2015; Graciani et al, 2011; Kovács et al, 2015). Thus, many satellite programs (pilots) running inside VMs branch out to monitor a set of computational user tasks that are continuously assigned by these frameworks following the master-slave scheme. The main motivations for establishing this network overlay within a cloud infrastructure are:

- to effectively characterise the deployed VMs by sending their real properties and current status to the master;
- to increase the robustness of the calculations;
- to enable compatibility with other legacy systems (such as grid services) and applications by checking and creating special configurations; and
- to reduce cloud complexity (especially in multi-cloud approaches) by directly using assigned resources and monitoring the user tasks.

In general, the pilot systems tested in real clouds have accomplished the last two items mentioned above. Nevertheless they do not properly deal with the first one and only partially with the second one. To improve the reliability of the calculations, the pilot frameworks must provide mechanisms that allow the inclusion of specialised scheduling algorithms, specifically devoted to these executions. For this purpose, the pilot system must not only support standardised APIs to code the algorithms, but also allow the personalised characterisation of provisioned resources as well as provide a reliable model to adapt them to the pilot behaviour. In this sense, the mentioned systems lack compatibility or adaptability. Furthermore,

[2] http://aws.amazon.com

the performance of most of them is unpredictable, which prevents coupling them with legacy self-schedulers, for example. For this reason, few abstraction models of pilot jobs have been proposed (Glatard and Camarasu-Pop, 2011; Luckow et al, 2012; Mościcki et al, 2011) that can be used for this purpose (Korkhov et al, 2009; Camarasu-Pop et al, 2013). These issues are properly tackled in this work.

3 Modelling task turnaroud with GWpilot

GWpilot (Rubio-Montero et al, 2015b) is a general-purpose pilot job framework embedded in the GridWay meta-scheduler (Huedo et al, 2007). The performance, adaptability and flexibility of the system have been demonstrated in grids, but its compatibility with cloud providers and its capacity of supporting third-party schedulers have been presented as future features.

Recently, the cloud support has been achieved (Rubio-Montero et al, 2015c). .For this purpose it was necessary to implement two new Information and Execution drivers able to manage cloud interfaces. This section is focused on explaining the capabilities related to cloud characterisation and modelling for scheduling, and it only compiles a general description of GWpilot and the cloud drivers in Subsection 3.1. For an extensive explanation of the design, behaviour and operation with the framework, reader should consult the references (Rubio-Montero et al, 2015c,b,a). In any case, the data extracted by the Information Driver from ISs of the infrastructures are the initial characterisation for every cloud provider. This information is used to establish the first filter based on the availability of hardware and virtual images, which is consequently used for virtual machine provisioning. Thus, the aspects needed to understand the scheduling support provided, are included in this section.

To support external schedulers the framework must provide a suitable model and a methodology to adapt a wide range of scheduling algorithms. According to the related work, this model should be based on task turnaround, while the mechanism of adaptation should allow incorporating personalised characterisation of resources without modifying legacy codes of the third party tools and applications. These two items, together with its practical demonstration, are the main contribution of this work. For this reason they are described in different Subsections 3.2 and 3.3.

3.1 The framework in cloud federations

The GWpilot framework counts on two main modules in addition to the pilots: the GWpilot Server and the Factory. The implementation of pilots is lightweight and without library dependencies, i.e. they can run on any kind of Linux OS. Thus, no especial configurations are needed to deploy the pilot overlay on cloud federations

According to the number and requirements of the tasks created by any application or third-party scheduler, the Factory automatically builds the necessary pilot jobs that will be executed in VMs. The description of these tasks can contain constraints to certain VM image identifiers related to cloud marketplaces, for example the *appdb.egi.eu* code for FedCloud images. Factory interprets them and includes those related ones into the pilot requirements.

The GridWay Scheduler will use the information dynamically updated by the GWcloud Information driver to select the most suitable cloud provider every time that will execute a task. For this purpose, the Scheduler takes account of the requirements set in the description of the pilot jobs, in a way similar to the one the described in (Rubio-Montero et al, 2015a). Then, the management of the VM creation and the pilot job execution are delegated to the GWcloud Execution driver. This module performs all the actions needed to run the pilotjob in a contextualisation script, interfacing with providers through the OCCI. Consequently, the pilots executed will enrol to GWpilot Server and the first level of scheduling (the resource provisioning) is successfully completed.

Therefore, users can run their legacy codes on GridWay as usual. The tasks created by these applications are also scheduled among the enrolled pilots. This constitutes the second level of scheduling (the task scheduling). Potentially, the combination of levels allows advanced scheduling techniques (Rubio-Montero et al, 2015b), although this work will be focused on allowing staking another scheduling layer on the top of the framework. To achieve this feature, the characterisation takes uppermost importance.

There are four characterisation mechanisms available in the framework. The default monitoring performed by pilots, the personal customisation based on characterisation tasks, the accounting performed by GridWay, and the information retrieved from cloud ISs. As commented, the latter is not accurate and is overcome by the other three, as it is explained with the model. In any case, this can be used as a first guide for scheduling and provisioning, and for this reason is described below. On the other hand, the customisation of tasks and the accounting are features used in the adaptation of the third-party scheduler presented in Section 4.

The GWcloud Information Driver

This new driver looks up for cloud providers in top BDIIs of one or multiple federations. Currently, the driver supports the EGI FedCloud, but it can be modified to directly use OCCI or AWS interfaces to work on a multi-cloud environment. Subsequently, the driver filters the information to dynamically notify GridWay about the characteristics of providers in which the user is authorised. Every provider found is included as an independent resource in the Host Pool. Thus, the information can be consulted by the user through the GridWay commands and it is shown as:

– The URI contact endpoint, the protocol, hypervisor and VIM releases, the maximum number of available cores, etc.
– Every OS template name (the *os_tpl*) and their *appdb.egi.eu* image identifier are compiled in a list of pairs and included as new tags.
– Every resource template (*resource_tpl*) is shown as a different queue, with its own characterisation: number of cores, memory, etc.

Thus, any developer can use these searches to constraint the matches to certain characteristics published by providers.

3.2 Simple turnaround model

The real turnaround time of each completed task is defined as the time difference between the moment when the task is queued in the framework for being dispatched (i.e. the application or the third-party scheduling tool make the submission) and the moment when the system notifies that it is completed. This value represents the CPU time consumed by the application when it is executed in the remote pilot plus the transfer times and the overhead introduced by the underlying middleware in its own scheduling.

Unlike other pilot systems used in cloud, related middleware overheads can be known *a priori* with the GWpilot configuration. Thus, when no failures are produced and enough amounts of pilots are available, the idealised turnaround (τ) is:

$$T = T_{sched} + T_{xfer} + T_{exec} \simeq t_{si/2} + t_{pi} + t_x + t_e = \tau \tag{2}$$

t_e and t_x maintain the same significance than T_{exec} and T_{xfer} in Equation 1. T_{exec} depends on the power of the resource selected, while T_{xfer} principally depends on the network and the amount of data transferred. Thus t_e as well as t_x should be estimated by a self-scheduler, but now GWpilot provides the characterisation of every pilot for this purpose. This is, the bandwidth, latency and power are published for every pilot. In addition, the tools needed to publish any specialised benchmarking of these or other performance aspects (for example, the disk throughput) are available in GWpilot.

On the other hand, the T_{sched} is the overhead related to the framework and task scheduling mechanisms. It is split into two types of process ($t_{si}+t_{pi}$), with different modelling behaviour. First, t_{pi} is the overhead related to pilot notifications and the capacity of the GWpilot Server to process them. When no failures or network cuts are registered, t_{pi} coincides with the pilot interval set in the GWpilot configuration.

$t_{si/2}$ stands for the time needed to obtain a suitable pilot to execute the task. GridWay Scheduler will prioritise some tasks over others and then will dispatches these tasks to pilots that accomplish their requirements. When tasks have identical requirements and enough amount of pilots are already appropriated by the pilot system, $t_{si/2}$ only depends on the elapsed time between schedules set in the GWpilot configuration (the half of $SCHEDULING\_INTERVAL$).

3.3 Methodology to incorporate third-party schedulers

With the model presented and the GWpilot features, it is possible to easily separate the characterisation, the application-level scheduling, the task scheduling, and the provisioning, while maintaining the control over the pilot submission and removing the penalties originated by self-made estimations. However, it could be of interest for the developer to keep some of his implemented procedures to better fit the needs of his application. Thus, a developer that plans to adapt any third-party scheduler to GWpilot should follow the following steps:

1. To identify the algorithm requirements (pre-conditions) that are related to characterisation, i.e. what qualifiers are necessary and what type of resources should be prioritised.

2. To check if GWpilot already offers procedures to accomplish these requirements and to value if they should substitute the possible legacy ones by:
   - Making a simple procedure to submit characterisation or customisation tasks to pilots whenever they were newly enrolled to the framework.
   - Configuring the GWpilot Factory or modifying the legacy mechanism to submit pilot jobs.
   - Removing the compilation of statistical data from tasks (or even pilots) if the accounting performed by the framework is enough.
3. To purge the third-party code from useless procedures that have been achieved by the previous techniques and reformulate the proposed algorithm if needed.
4. To adjust the algorithm to establish equilibrium between spent time and profitable execution time taking into account the turnaround model and accordingly set the GWpilot configuration.

Obviously, not all these items must be completed to obtain a useful adaptation. Many developers do not like to modify legacy codes to take advantage of pilots. However, to show the simplicity of the adaptation although it is the theoretically most expensive approach, as well as to profit from all the GWpilot features, the methodology is completely followed in this work.

4 Resilient executions of MC codes

To demonstrate how the proposed model and methodologies are suitable for incorporating external scheduling algorithms, even for those already included in legacy software, a good example should be the adaptation of a self-scheduler to GWpilot. For this purpose, the Montera framework developed by Rodríguez-Pascual et al (2013) has been used. Therefore, the *Dynamic Trapezoid Self-Scheduling*(DyTSS) algorithm will be utilised as a proof of concept to show how a loop-scheduler can be adapted to GWpilot.

It could seem that as the framework relies on GridWay, the procedure presented is only bounded to frameworks that use this platform (Díaz et al, 2009; Tomás et al, 2012). However, the binding with GridWay is mainly based on DRMAA and subsequently, other applications implemented with standards (DRMAA and OGSA-BES) should be straightforwardly adapted too. Moreover, other frameworks that use different grid schedulers or batch managers can be also easily adapted by substituting some commands, because getting information about resources and managing jobs is very simple with GWpilot.

4.1 Adaptive sample-based algorithm

DyTSS (Rodríguez-Pascual et al, 2013) is a loop-based algorithm (Díaz et al, 2009) that differentiates from other approaches such as TSS (Tzen and Ni, 1993) or GTSS (Herrera, 2009) in its dynamic nature and in its focus on managing MC codes. The enhanced version of DyTSS used in this work is described in Algorithm 1. It shows the pseudo-code resultant of observing more properly the mathematical basis and purging the functions without relationship to the algorithm itself. Moreover, the main advance is the inclusion of the minimum (L) and the maximum (M) chunk-size limits as external configuration parameters. With them, users can adjust better its behaviour on the network overlay created by pilot jobs by limiting the overhead of shorter tasks and the execution time of longer ones.

Dynamic algorithms usually need to continuously calculate the suitable execution time in every provider. The objective is to fit the workload partitioning to the estimated status of the infrastructure. For this reason it is usually necessary to benchmark the infrastructure performance and to profile the application. These actions are also necessary when pilot jobs are used. In this sense, some general-purpose algorithms (Korkhov et al, 2009), and even ones devoted to MC (Camarasu-Pop et al, 2013), are based on calculating the minimal real execution time in every resource to reduce the overhead percentage of every task to a certain threshold.

With DyTSS the approach is completely different as L and M are preset by the user. These values can be previously calculated taking into account the mentioned overheads, but the algorithm will never modify those values. The approach of DyTSS is to reduce overheads by adjusting the first executions as much as possible to M, but straighten turnarounds. Then, it progressively decrease the chunk size to L, overcoming the influence of failed jobs in the makespan. The result is a improved and resilient calculation with reduced makespan.

The algorithm calculates the number of samples (s_j) to submit to every available resource (r_j) belonging to a characterised infrastructure (R). For this purpose, the current power of the infrastructure is estimated by linear regression of the ordered performances from (r_j, s_j) pairs. The ordinate in the origin ($n_{1/2}$) of the obtained straight-line corresponds to the half-performance of the infrastructure for this distribution of tasks (Montero et al, 2006). This value determines the variable component (F) of the maximum chunk size ($F + L <= M$) for the current loop stage. As commented, L and M are constant, and consequently the number of samples is always within the interval ($L \cdots M$), but the turnaround will be different for every match (r_j, s_j). Thus, the procedure is repeated until no improvement is obtained for the sample distribution.

Note that (F) should decrease with the number of samples (S) as the simulation is being accomplished ($\text{end}(s_j)$) if not a new more powerful ($\text{new}(r)$) resource is discovered. This assures cutting final execution tail due to remaining and standalone tasks.

4.2 Characterisation and adaptation

Montera supports by default two characterisation modes: it estimates the available number, power and bandwidth of the slots offered by the infrastructure at a certain time, as well as the computational needs of the application. The main difference among other frameworks is how those parameters are estimated. In particular, the system:

- Benchmarks every provider by means of submitting a testing job that measures the CPU performance in manageable units (for example, whetstones (Curnow and Wichmann, 1976)) as well as the bandwidth.
- Then, averages the aforementioned benchmarks with every real execution of the real application, taking also into account the time needed for staging its input and output files.

Algorithm 1: DyTSS.

```
Require: $R \equiv [r_{max}...r_{min}]$ (list of estimated available resources)
Require: $L$ (minimum sample-chunk)
Require: $M$ (maximum sample-chunk)
Require: $S$ (number of required samples)
$task\_list \leftarrow \{\emptyset\}$
for $r_j \in R$ do
    $task\_list \leftarrow task\_list \cup \{(r_j, L)\}$
$free\_res\_list \leftarrow R$
while $S > 0$ do
    for $i = 1 \rightarrow 100$ do
        $old\_task\_list \leftarrow task\_list$
        $n_{1/2} \leftarrow linear\_fit(old\_task\_list)$
        $F \leftarrow S/4 \cdot n_{1/2}$
        $task\_list \leftarrow \{\emptyset\}$
        for $r_j \in R$ do
            $f_{rel} \leftarrow T(r_{max}, L)/T(r_j, L)$
            $s \leftarrow minimum(M, F \cdot f_{rel} + L)$
            $task\_list \leftarrow task\_list + \{(r_j, s)\}$
        if $task\_list \equiv old\_task\_list$ then break
    for $r_j \in free\_res\_list$ do
        $submit((r_j, s_j) \in task\_list)$
    $free\_res\_list \leftarrow \{\emptyset\}$
    for $end(s_j), (r_j, s_j) \in task\_list$ do
        if $OK(s_j)$ then
            $task\_list \leftarrow task\_list - \{(r_j, s_j)\}$
            $S \leftarrow S - \{s_j\}$
            $free\_res\_list \leftarrow free\_res\_list \cup \{r_j\}$
    for $new(r)$ do
        if $T(r_{min}) > T(r)$ then
            $R \leftarrow R \cup \{r\}$
            $free\_res\_list \leftarrow free\_res\_list \cup \{r\}$
```

– Estimates the slot availability and reliability of every provider averaging the number of failed attempts and the number of failed tasks.

However, the characterisation mechanisms implemented in Montera are oriented to submit jobs directly to grid sites. For this purpose, replication is used for testing the slot availability, and an exhaustive accounting is performed inspecting job outputs.

However, these techniques are unnecessary in the network overlay created over the cloud resources. GWpilot features can be fully configured to satisfy all provisioning matters without replication, as well as characterisation tasks can be submitted to pilots and the accounting of GridWay can be finally used, avoiding the necessity of compiling statistics. In this sense, the benefits of continuously testing the providers can now be achieved by the correct configuration of GWpilot Factory and the banning feature, as has been explained in (Rubio-Montero et al, 2015c).

It is noteworthy to mention that not only the allowed number of VMs in every cloud provider is variable, even the hardware provided by each resource is also so because it can be composed by different kinds of nodes and the provider can be overbooked. Therefore, the basis of the characterisation should rely on the same mathematical basis used for grid, but adapted to the turnaround model proposed for pilot jobs.

4.2.1 Application profiling and resource benchmarking

Pilots provisioned are seen by Montera as suitable resources. However, these pilots have to be characterised before being profited. On the other hand, the application has to be profiled according to the benchmarks used for pilots to enable the matchmaking among them.

Therefore, the operation with a new MC code starts analysing its computational needs. For this purpose, it is executed on a set of pilots progressively growing the number of samples, so its requirements in terms of CPU consumption and data movement can be profiled. The obtained parameters are the *Constant effort*, C_{eff} (the effort necessary to execute the constant part of the application, like compiling code, input pre-processing, output post-processing, etc.), and the *Sample effort*, S_{eff} (the effort necessary to simulate a single *sample*). These efforts can be measured in the performance units provided by any general-purpose CPU benchmark tool to increase the suitability of the solution for a wide range of computational requirements. In this work, Whetstone (Curnow and Wichmann, 1976) has been selected.

To benchmark the pilot, a characterisation task is submitted. Unlike the profiling tasks, the interest is now focused on its *performance* (P) and its *bandwidth* (BW). To obtain the information, the benchmark tool is executed again and a big file is copied. This analysis is always performed when a new pilot is detected and the results are published as an additional tag for this pilot. To preserve the accurateness of the monitoring, the values published are updated as any conventional task that belongs to the real calculation ends, as will be explained below.

4.2.2 New turnaround estimation

Using the Equation 2, the aforementioned parameters are used to estimate the turnaround time (T) of executing an arbitrary number of simulations (s) of a given application on any remote resource ($r_j \in R$), the calculation of which requires and generates certain data (D) to transfer:

$$\begin{aligned} T(r_j, s) \simeq \tau(p_j, s) = \\ = t_{si/2} + t_{pi} + \frac{D}{BW_{p_j}} + \frac{C_{eff} + s \cdot S_{eff}}{P_{p_j}} \end{aligned} \tag{3}$$

However, this model loses precision and is unreliable if benchmarking parameters are not updated during the execution of an application, as they vary through the time. To avoid this, every successful execution of a MC task is analysed to re-calculate the parameters, so knowledge about the infrastructure is enhanced in real time with minimum computational effort. Moreover, DyTSS can receive now the exact benchmark of every slot effectively appropriated with the GWpilot framework. This increases the efficiency of the algorithm. Additionally, the overheads introduced are reduced because DyTSS has to wait neither for benchmarks, nor for storing statistical data.

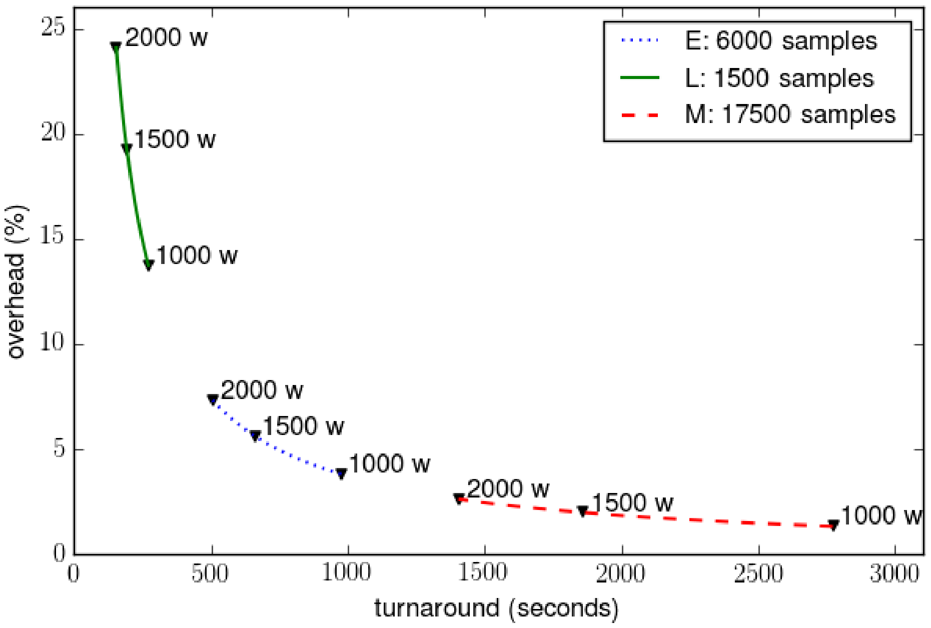

Figure 1 Overhead with respect to expected turnaround (Equation 3) according to the number of Nagano's samples per task and the GWpilot configuration ($t_{si/2} = 5$ s, $t_{pi} = 30$ s). An idealised infrastructure with $D/BW_{p_j} = 2$ s and without failures is considered, but cloud providers offer resources with benchmarked power (P) from 1000 to 2000 *whetstones* (w).

Therefore, as the accurateness of the model is preserved through the changes in the availability of resources, it can be used by the adaptive algorithm to determine the global performance of the virtual infrastructure provisioned and to consequently adapt the number of samples (s) to submit and where to do it.

5 Experiments

The objective is to evaluate the effectiveness of the model to incorporate advanced scheduling algorithms into the late-binding approach offered by GWpilot. For this purpose, the execution of a MC application is performed with and without stacking Montera to the GWpilot/GWcloud framework in the EGI FedCloud infrastructure.

5.1 Proposed tests

Nagano (Vélez., 2011) is a MC application based on the experiments of Nagano et al (2003, 2004). The code is devoted to simulate fluorescence emissions and the energy deposited by electrons inside an observation volume of the desired proportions. Thus, Nagano lies on the area of astroparticle physics, helping on the research of the origin and propagation of ultra-high-energy cosmic rays. The calculation for a single electron takes only a fraction of second, but for a real world use case, a complete simulation comprises several millions of electrons, for example

$2 \cdot 10^7$. In this sense, its execution can be characterised following the profiling explained in subsection 4.2, resulting in $C_{eff} = 375.07$ w and $S_{eff} = 156.26$ w, being w *whetstone* units. The inputs and the application itself take 500 KB and output files require only a few KB, then the stage-in and -out process is really ballasted by the negotiation of the connection, lasting less than 2 seconds in current research or business networks.

Figure 1 shows the estimation of the overhead evolution for a task when it is executed on certain provisioned pilot, using the turnaround model proposed in this work and taking into account that resources offered by current cloud providers usually ranges from 1,000 w to 2,000w. Each number of samples used coincides with the ones selected for the proposed tests. These are the limits L and M for DyTSS and the fixed number of samples E for an equal-sample-size distribution. As can be seen, these values are not arbitrary chosen. The overheads of the calculation based on equal-sized tasks are theoretically bounded into a 4-7%, according to the turnaround model. In addition, the calculation should be few influenced by the failures of tasks due to their short duration (between 10 and 20 minutes). On the other hand, DyTSS has to reduce the makespan by dealing with very short and long tasks, i.e. achieving equilibrium among overheads, failures and turnaround to increase the production of samples.

Currently, the real availability of resources is very limited in FedCloud. Although a virtual image widely deployed in providers were used (Rubio-Montero et al, 2015c), the maximum number of VMs provisioned varies from just a few tens to one hundred fifty, distributed among about seven reliable providers. This issue constraints the tests performed and the comparison to the results obtained in previous works (Rubio-Montero et al, 2015d) following the early-binding approach. Consequently, two types of tests have been performed. The first one is the calculation of $2 \cdot 10^7$ samples divided in 3,333 tasks with fixed E size. The experiment is repeated three times to study the sample production according to the number of pilots provisioned. For the last one, Montera was stacked to GWpilot/GWcloud to perform the same calculation another three times, but using the aforementioned limits L and M. The idea is to use the conclusions made in first experiment to analyse the reliability added by DyTSS to the calculation, independently of the number of pilots. These experiments have been launched at the same hour in different days. For this purpose, a desktop machine with one i3-530 processor (2 cores, 2.93 GHz) was used to support GWpilot and GWcloud. The configuration parameters were identical to the ones used for the estimation in Figure 1: an scheduling interval of 10 s ($t_{si/2} = 5$ s) and pilot interval (t_{pi}) of 30 s. Additionally, the GWpilot/GWcloud framework are allowed for managing a maximum of 200 pilots (running on 200 VMs of a widely deployed marketplace image[3]).

5.2 Results

Regarding how the framework behaves when Montera is not stacked, it can be seen in Figure2 that the number of calculated samples increases as the number of pilots do, as expected. Slopes are shown during the longer interval in which

[3] https://appdb.egi.eu/store/vo/image/de355bfb-5781-5b0c-9ccd-9bd3d0d2be06

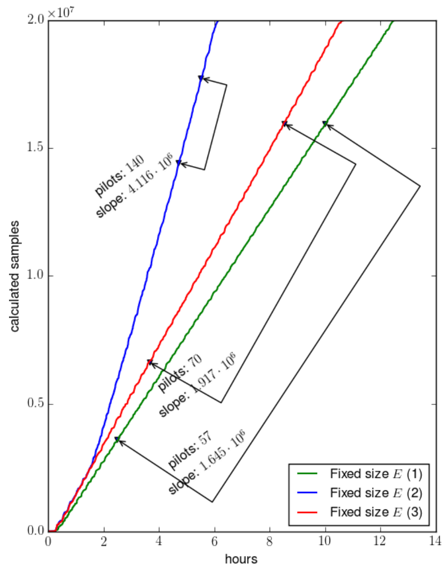

Figure 2 Nagano production using tasks with fixed size E of 6000 particles (without DyTSS).

the number of pilots is maintained constant in the experiment. In this sense, a conclusion can be found if how the slope (i.e. number of calculated samples) increases with the number of jobs is analysed. Fitting to a new linear curve the number of pilots and the slope, it is found that the number of calculated samples increases in a factor of $\sim$ 32 as the number of pilots do (this fitting presents a correlation factor r of 0.999 and a R^2 coefficient of 0.998). On the other hand, the linear regression between the interval points where the pilots remain stable coincides with the performance of the provisioned infrastructure using this sample distribution (Montero et al, 2006). The heuristic of DyTSS is based on calculating this performance for different distributions, and therefore it can be used to verify if Montera achieves better results than a fine equal-sized choice. It is noteworthy to mention that the type of provisioned resources varies among experiments, but the slopes are similar when the number of pilots is maintained stable.

As can be seen in Figure 3, when Montera is stacked to GWpilot/GWcloud , the coupled tool outperforms the simple GWpilot/GWcloud framework in the long term, although the latter had provisioned whole pilots previously the experiment starts. To find this conclusion, the productivity of the experiment with fixed size E has been estimated by calculating the slope for a number of pilots equivalent to the real average of pilots provisioned with Montera. In the beginning, the former experiment executes better as its lineal behaviour starts calculating samples more quickly. During those initial hours, the stacked system presents an exponential behaviour as the proper provision takes longer, but from one point on (around 3 hours and 45 minutes), it is able to calculate more samples. The result is that experiments without Montera would last $\sim$ 8-13% more.

Figure 3 also shows the behaviour of the DyTSS algorithm. With DyTSS, the first tasks to be executed are bigger than the last ones. This aims to reduce the overheads during the most part of the calculation, while limiting the influ-

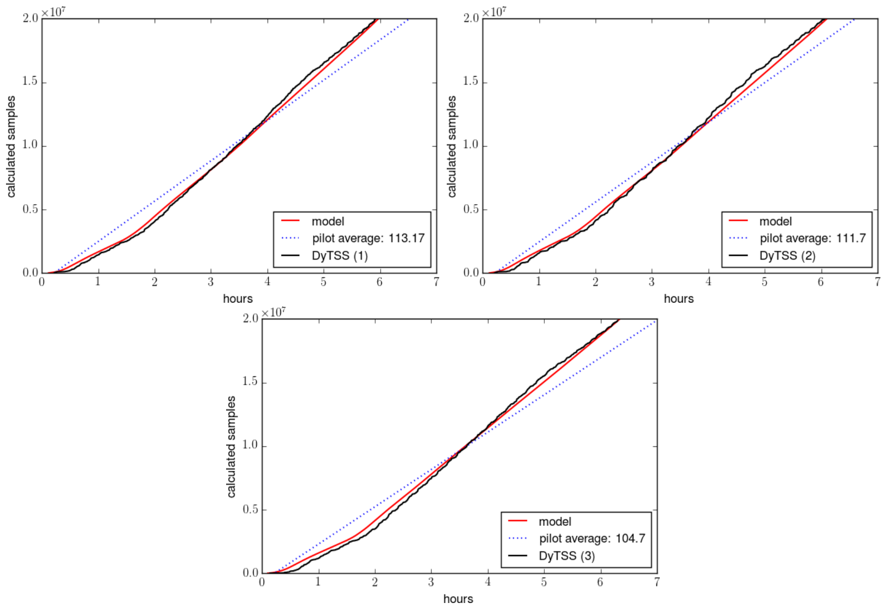

Figure 3 Nagano production with DyTSS algorithm.

ence of failed tasks close to the end. However, this fact apparently decreases the productivity (measured as number of successfully completed samples) at the beginning, because it takes quite a long time until these larger tasks start returning the first results. . The only way to describe this issue is drawing the sample production as the system were working with a task size between L and M, for example E. As the number of pilots and their benchmark average have been monitored every minute, the turnaround model can be used for this purpose. Thus, in Figure 3 it is also depicted the proposed mathematical model (red line). Using this reference it can be seen that DyTSS works first under that line and then above it. Furthermore, it clearly demonstrates that the proposed model fits the real execution of the pilots in a real cloud infrastructure as FedCloud is. This fact is key to foresee the behaviour of a cloud infrastructure for scheduling and provisioning resources in advance.

6 Conclusions

To increase the computational efficiency in distributed computing infrastructures, a new model has been presented. It can be used to properly exploit the heterogeneity and the real availability of resources belonging to multiple cloud providers. For this purpose, the model has been included in the GWpilot framework that has extended its capabilities to cloud environments. Unlike other pilot systems, the proposed framework is capable of supporting diverse scheduling algorithms, even provided by third-party tools. This is so because the personalised characterisation that those algorithms require is possible, a fact that overcomes their lack of trustworthiness in the information provided by cloud services. The suitability of the approach has been demonstrated with a legacy self-scheduler (Montera) specialised on performing reliable MC executions on distributed resources based on heuristics, which has been stacked and tested on the EGI FedCloud infrastructure with the Nagano legacy application. First results on this cloud infrastructure, which is in production status, demonstrate that the cloud system presents a linear behaviour of calculated samples with the average number of pilots provisioned. Additional tests have also compared how the framework behaves with and without the Montera self-scheduler stacked. The analysis demonstrates that the former system improves the performance of the latter one due to its better fitting of the real characteristics of cloud resources. In this sense, the model proposed in this work perfectly matches the real production of the legacy self-scheduler when has been stacked onto the pilot system. The overhead produced with respect to the expected turnaround has been analysed as well.

Compliance with Ethical Standards

Funding: this work was supported by the COST Actions BETTY (IC 1201) and NESUS (IC1305) and partially funded by the Spanish Ministry of Economy and Competitiveness project CODEC2 (TIN2015-63562-R).

Conflict of Interest: authors declare that they have no conflict of interest.

Ethical Approval: this article does not contain any studies with human participants or animals performed by any of the authors.

References

Abdullah M, Othman M (2013) Cost-based Multi-QoS Job Scheduling Using Divisible Load Theory in Cloud Computing. In: International Conference on Computational Science (ICCS 2013), Elsevier, Barcelona, Spain, Procedia Computer Science, vol 18, pp 928–935, DOI 10.1016/j.procs.2013.05.258

Aceto G, Botta A, de Donato W, Pescapè A (2013) Cloud monitoring: A survey . Computer Networks 57(9):2093–2115, DOI 10.1016/j.comnet.2013.04.001

Anastasi GF, Carlini E, Coppola M, Dazzi P (2014) BROKAGE: A Genetic Approach for QoS Cloud Brokering. In: 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014), Alaska. USA, pp 304–311, DOI 10.1109/CLOUD.2014.49

Andreozzi S, Burke S, Ehm F, Field L, Galang G, Konya B, Litmaath M, Millar P, Navarro JP (2009) GLUE Specification v. 2.0. URL http://www.ogf.org/documents/GFD.147.pdf, GFD 147

Babu D, Venkata P (2013) Honey bee behavior inspired load balancing of tasks in cloud computing environments. Applied Soft Computing 13(5):2292 – 2303, DOI 10.1016/j.asoc.2013.01.025

Bala A, Chana I (2015) Autonomic fault tolerant scheduling approach for scientific workflows in Cloud computing. Concurrent Engineering 23(1):27–39, DOI 10.1177/1063293X14567783

Camarasu-Pop S, Glatard T, da Silva RF, Gueth P, Sarrut D, Benoit-Cattin H (2013) Monte Carlo simulation on heterogeneous distributed systems: A computing framework with parallel merging and checkpointing strategies. Future Generation Computer Systems 29(3):728–738, DOI 10.1016/j.future.2012.09.003

Chiu CF, Hsu S, Jan SR, Chen JA (2014) Task scheduling based on load approximation in cloud computing environment. In: Future Information Technology, Lecture Notes in Electrical Engineering, vol 309, Springer Berlin Heidelberg, pp 803–808, DOI 10.1007/978-3-642-55038-6_122

Ciuffoletti A (2014) A Simple and Generic Interface for a Cloud Monitoring Service. In: CLOSER 2014 Proceedings of the 4th International Conference on Cloud Computing and Services Science, SCITEPRESS – Science and Technology Publications, Barcelona, Spain, pp 143–150

Curnow HJ, Wichmann BA (1976) A synthetic benchmark. The Computer Journal 19(1):43–49, DOI 10.1093/comjnl/19.1.43

Díaz J, Reyes S, no AN, noz Caro CM (2009) Derivation of self-scheduling algorithms for heterogeneous distributed computer systems: Application to internet-based grids of computers. Future Generation Computer Systems 25(6):617–626, DOI 10.1016/j.future.2008.12.003

Foster I, Zhao Y, I Raicu I, Lu S (2008) Cloud Computing and Grid Computing 360-Degree Compared. In: Grid Computing Environments Workshop (GCE '08), IEEE, Austin, TX, USA., pp 1 – 10, DOI 10.1109/GCE.2008.4738445

Garey M, Johnson D (1979) Computers and Intractibility: A guide to the Theory of NP-completeness. W. H. Freeman&Co, New York

Glatard T, Camarasu-Pop S (2011) A model of pilot-job resource provisioning on production grids. Parallel Computing 37(10–11):684–692, DOI 10.1016/j.parco.2011.04.001

Gómez-Iglesias A, Vega-Rodríguez MA, Castejón F, Morales-Ramos E, Cárdenas-Montes M, Reynolds JM (2010) Grid-based metaheuristics to improve a nuclear fusion device. Concurrency Computat: Pract Exper 22(11):1476–1493, DOI 10.1002/cpe.1497

Graciani R, Casajús A, Carmona A, Fifield T, Sevior M (2011) Belle-DIRAC Setup for Using Amazon Elastic Compute Cloud. Journal of Grid Computing 9(1):65–79, DOI 10.1007/s10723-010-9175-7

Grozev N, Buyya R (2014) Inter-Cloud architectures and application brokering: taxonomy and survey. Softw: Pract Exper 44:369–390, DOI 10.1002/spe.2168

Herrera J (2009) Programming Model for Grid Computing Infrastructures. (in Spanish). PhD thesis, Universidad Complutense de Madrid, Madrid, Spain

Huedo E, Montero RS, Llorente IM (2007) A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services. Future Generation Computer Systems 23(2):252–261 DOI 10.1016/j.future.2006.07.013

Korkhov VV, Mościcki JT, Krzhizhanovskaya VV (2009) Dynamic workload balancing of parallel applications with user-level scheduling on the Grid. Future Generation Computer Systems 25(1):28–34, DOI 10.1016/j.future.2008.07.001

Kovács J, Marosi AC, Visegrádi A, Farkas Z, Kacsuk P, Lovas R (2015) Boosting gLite with cloud augmented volunteer computing. Future Generation Computer Systems 43–44:12–23 DOI 10.1016/j.future.2014.10.005

Lu K, Yahyapour R, Wieder P, Yaqub E, Abdullah M, Schloer B, Kotsokalis C (2016) Fault-tolerant service level agreement lifecycle management in clouds using actor system. Future Generation Computer Systems 54: 247–259 DOI 10.1016/j.future.2015.03.016

Lucas-Simarro JL, Moreno-Vozmediano R, Montero RS, Llorente IM (2015) Cost optimization of virtual infrastructures in dynamic multi-cloud scenarios. Concurrency Computat: Pract Exper 27(9):2260–2277, DOI 10.1002/cpe.2972

Luckow A, Santcroos M, Merzky A, Weidner O, Mantha P, Jha S (2012) P*: A model of pilot-abstractions. In: 8th IEEE International Conference on E-Science (e-Science 2012), Chicago, USA, pp 1–10, DOI 10.1109/eScience.2012.6404423

Luckow A, Santcroos M, Zebrowski A, Jha S (2015) Pilot-Data: An abstraction for distributed data. Journal of Parallel and Distributed Computing 7980:16–30, DOI 10.1016/j.jpdc.2014.09.009

Mhashilkar P, Tiradani A, Holzman B, Larson K, Sfiligoi I, Rynge M (2014) Cloud Bursting with GlideinWMS: Means to satisfy ever increasing computing needs for Scientific Workflows. In: 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013), IOP Publishing, Journal of Physics: Conference Series, vol 513, p 032069, DOI 10.1088/1742-6596/513/3/032069

Mohamed M, Amziani M, Belaïd D, Tata S, Melliti T (2015) An autonomic approach to manage elasticity of business processes in the cloud. Future Generation Computer Systems 50:49 – 61, DOI 10.1016/j.future.2014.10.017

Montero R, Huedo E, Llorente I (2006) Benchmarking of high throughput computing applications on Grids. Parallel Computing 32(4):267–279, DOI 10.1016/j.parco.2005.12.001

Moon YH, Youn CH (2015) Multihybrid job scheduling for fault-tolerant distributed computing in policy-constrained resource networks. Computer Networks 82:81 – 95, DOI 10.1016/j.comnet.2015.02.030

Moreno-Vozmediano R, Montero RS, Llorente IM (2012) IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. Computer 45(12):65–72, DOI 10.1109/MC.2012.76

Mościcki JT (2011) Understanding and Mastering Dynamics in Computing Grids: Processing Moldable Tasks with User-Level Overlay. PhD thesis, Universiteit van Amsterdam, Nederlands

Mościcki JT, Lamannaa M, Bubak M, Sloot PMA (2011) Processing moldable tasks on the Grid: Late job binding with lightweight user-level overlay. Future Generation Computer Systems 27(6):725–736, DOI 10.1016/j.future.2011.02.002

Nagano M, Kobayakawa K, Sakaki N, Ando K (2003) Photon yields from nitrogen gas and dry air excited by electrons. Astroparticle Physics 20(3):293 – 309, DOI 10.1016/S0927-6505(03)00192-0

Nagano M, Kobayakawa K, Sakaki N, Ando K (2004) New measurement on photon yields from air and the application to the energy estimation of primary cosmic rays. Astroparticle Physics 22(3–4):235 –248, DOI 10.1016/j.astropartphys.2004.08.002

Nesmachnow S, Cancela H, Alba E (2010) Heterogeneous computing scheduling with evolutionary algorithms. Soft Computing 15(4):685–701, DOI 10.1007/s00500-010-0594-y

Panda SK, Gupta I, Jana PK (2015) Allocation-aware Task Scheduling for Heterogeneous Multi-cloud Systems. In: 2nd International Symposium on Big Data and Cloud Computing Challenges (ISBCC '15), Chennai, India, Procedia Computer Science, vol 50, pp 176 – 184, DOI 10.1016/j.procs.2015.04.081

Parák B, Šustr Z, Feldhaus F, Kasprzakc P, Srbac M (2014) The rOCCI Project: Providing Cloud Interoperability with OCCI 1.1. In: International Symposium on Grids and Clouds (ISGC), Taipei, Taiwan, SISA PoS, pp 1–15

Pinedo M (2005) Planning and Scheduling in Manufacturing and Services. Springer Series in Operations Research, Springer, New York, DOI 10.1007/b139030

Sajid M, Razaa Z (2015) Turnaround Time Minimization-Based Static Scheduling Model Using Task Duplication for Fine-Grained Parallel Applications onto Hybrid Cloud Environment. IETE Journal of Research DOI 10.1080/03772063.2015.1075911, In press. Available Online.

Rodríguez-Pascual M, Mayo-García R, Llorente IM (2013) Montera: a framework for efficient execution of Monte Carlo codes on grid infrastructures. Computing and Informatics 32(1):113–144

Rubio-Montero AJ, Castejón F, Huedo E, Mayo-García R (2015a) A novel pilot job approach for improving the execution of distributed codes: application to the study of ordering in collisional transport in fusion plasmas. Concurrency Computat: Pract Exper 27(13):3220–3244, DOI 10.1002/cpe.3301

Rubio-Montero AJ, Huedo E, Castejón F, Mayo-García R (2015b) GWpilot: Enabling multi-level scheduling in distributed infrastructures with GridWay and pilot jobs. Future Generation Computer Systems 45:25–52, DOI 10.1016/j.future.2014.10.003

Rubio-Montero AJ, Huedo E, Mayo-García R (2015c) User-guided provisioning in federated clouds for distributed calculations. In: Workshop on Adaptive Resource Management and Scheduling for Cloud Computing (ARMS-CC 2015), San Sebastián, Spain, Lecture Notes in Computer Science, vol. 9438, pp 60–77, DOI 10.1007/978-3-319-28448-4_5

Rubio-Montero AJ, Rodríguez-Pascual MA, Mayo-García R (2015d) Evaluation of an adaptive framework for resilient Monte Carlo executions. In: 30th ACM/SIGAPP Symposium On Applied Computing (SAC'15), Salamanca, Spain, pp 448–455, DOI 10.1145/2695664.2695890

Saleh A (2013) An efficient grid-scheduling strategy based on a fuzzy matchmaking approach. Soft Computing 17(3):467–487, DOI 10.1007/s00500-012-0920-7

Sheikhalishahi M, Wallace R, Grandinetti L, Vázquez-Poletti JL, Guerriero F (2015) A multi-dimensional job scheduling. Future Generation Computer Systems DOI 10.1016/j.future.2015.03.014, Available Online.

Shie MR, Liu CY, Lee YF, Lin YC, Lai KC (2014) Distributed Scheduling Approach Based on Game Theory in the Federated Cloud. In: International Conference on Information Science and Applications (ICISA 2014), IEEE CS Press, Seoul, South Corea, pp 1–4, DOI 10.1109/ICISA.2014.6847388

Smanchat S, Viriyapant K (2015) Taxonomies of workflow scheduling problem and techniques in the cloud. Future Generation Computer Systems 52:1–12, DOI 10.1016/j.future.2015.04.019

Snyder B, Ringenberg J, Green R, Devabhaktuni V, Alam M (2015) Evaluation and design of highly reliable and highly utilized cloud computing systems. Journal of Cloud Computing: Advances, Systems and Applications 4(11):1–16, DOI 10.1186/s13677-015-0036-6

Tao F, Feng Y, Zhang L, Liao T (2014) Clps-ga: A case library and pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling. Applied Soft Computing 19:264 – 279, DOI 10.1016/j.asoc.2014.01.036

Tomás L, Caminero AC, Rana O, Carrión C, Caminero B (2012) A GridWay-based autonomic network-aware metascheduler. Future Generation Computer Systems 28(7):1058–1069, DOI 10.1016/j.future.2011.08.019

Tordsson J, Montero RS, Moreno-Vozmediano R, Llorente IM (2012) Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Generation Computer Systems 28(2):358–367, DOI 10.1016/j.future.2011.07.003

Tzen TH, Ni LM (1993) Trapezoid self-scheduling: a practical scheduling scheme for parallel compilers. IEEE Transactions on Parallel and Distributed Systems 4(1), DOI 10.1109/71.205655

Vélez JR (2011) Analysis of the air fluorescence induced by electrons for application to cosmic ray detection. PhD thesis, Universidad Complutense de Madrid, Madrid, Spain

Wang X, Wang Y, Cui Y (2016) An energy-aware bi-level optimization model for multi-job scheduling problems under cloud computing. Soft Computing 20(1):303–317, DOI 10.1007/s00500-014-1506-3

Xu B, Peng Z, Xiao F, Gates A, Yu JP (2015) Dynamic deployment of virtual machines in cloud computing using multi-objective optimization. Soft Computing 19(8):2265–2273, DOI 10.1007/s00500-014-1406-6

Yangui S, Marshall IJ, Laisne JP, Tata S (2014) CompatibleOne: The Open Source Cloud Broker. J Grid Computing 12(1):93–109, DOI 10.1007/s10723-013-9285-0

Zanikolas S, Sakellariou R (2005) A taxonomy of grid monitoring systems. Future Generation Computer Systems 21(1):163–188, DOI 10.1016/j.future.2004.07.002

Zhani M, Boutaba R (2015) Survivability and Fault Tolerance in the Cloud, John Wiley & Sons, Inc., pp 295–308. DOI 10.1002/9781119042655.ch12