Computing and Informatics, Vol. 32, 2013, 113–144

MONTERA: A FRAMEWORK FOR EFFICIENT EXECUTION OF MONTE CARLO CODES ON GRID INFRASTRUCTURES

Manuel Rodríguez-Pascual, Rafael Mayo-García

CIEMAT
Avda Complutense, 22, 28040 Madrid (Spain)
e-mail: {manuel.rodriguez, rafael.mayo}@ciemat.es

Ignacio M. Llorente

DSA-Research.org. Universidad Complutense
C/ Prof. José García Santesmases s/n, 28040 Madrid (Spain)
e-mail: llorente@dacya.ucm.es

Communicated by Isabel Campos Plasencia

Abstract. The objective of this work is to improve the performance of Monte Carlo codes on Grid production infrastructures. To do so, the codes and the grid *sites* are characterized with simple parameters to model their behaviors. Then, a new performance model for grid infrastructures is proposed, and an algorithm that employs this information is described. This algorithm dynamically calculates the number and size of tasks to execute on each *site* to maximize the performance and reduce *makespan*. Finally, a newly developed framework called Montera is presented. Montera deals with the execution of Monte Carlo codes in an unattended way, isolating the complexity of the problem from the final user. By employing two fusion Monte Carlo codes as example cases, along with the described characterizations and scheduling algorithm, a performance improvement up to 650 % over current best results is obtained on a real production infrastructure, together with enhanced stability and robustness.

Keywords: Scheduling, task grouping, grid computing, Monte Carlo, performance

Mathematics Subject Classification 2010: 68W15, 68W40, 65C05

1 INTRODUCTION

Monte Carlo (MC) codes rely on the assumption that the behavior of complex phenomena can often be modeled by the execution of multiple simple simulations that employ random or pseudo-random numbers to compute their results. Thanks to this particularity, MC codes are widely used in various fields of computing simulations, solving problems where it is unfeasible or impossible to compute an exact result with a deterministic algorithm. The modularity of MC codes also simplifies the execution of distributed environments, where task distribution and synchronization is not trivial. MC codes have been successfully employed in fields as diverse as radiological medicine [4, 36], economics [5], finances [24, 8], environmental research [37], high energy physics [21, 49] or aerospace engineering [46], just to mention a few.

The MC codes considered for this work are the ones consisting on sets of independent simulations, whose architecture allows a straightforward parallelization. Other kind of codes such as Random Walks [42] or Markov's Chains [39, 15] are beyond the scope of this work. Thus, from now on, references to "Monte Carlo" should be understood as references to this subset of Monte Carlo-based applications.

Because of the high demand many scientific applications place on resources – MC codes among them – grid computing has emerged as a powerful platform for facing new and more ambitious problems. Grid computing has enabled the scientific community to have easier access to large computing resources beyond supercomputers. The nature of MC codes makes their parallelization straightforward, and they have been successfully ported to the grid on multiple occasions. However, the execution of MC codes still needs improvement to make the most of their flexibility in combination with the dynamicity of grid infrastructures, the characteristics of which have either not been taken into account or have only been dealt with in ideal environments.

Due to the particularities of this kind of infrastructure, the efficient execution of distributed applications is far from trivial. As detailed in the Related Works section, enormous effort has been put into this area, with many different approaches focusing on different concepts of "Grid Infrastructure", the particular characteristics of the application to be executed or the available and published information. Still, there is no specific tool for executing MC based applications on the grid, even though MC is one of the most widely employed paradigms. In this work, the problem is tackled through the creation of a framework specifically designed to fit the requirements of MC applications.

For this purpose, an innovative strategy that relies on three complementary tools is followed. These tools are employed to gather information from different sources and to distribute the samples to be simulated among the available resources. The three tools are as follows:

- information about the grid infrastructure based on static and dynamic data collected from past executions and the current status;

- an automatic analysis and characterization of the application to execute to model its behavior; and,
- the employment of a new scheduling algorithm.

To carry on this task, a number of contributions are presented on this paper.

First, an innovative characterization of MC codes and grid infrastructures with a small set of simple parameters is proposed. This way, the behavior of the application and infrastructure performance on any given moment can be accurately predicted, thus providing very valuable information to the scheduler.

The next step consisted in the design of a new scheduling algorithm called *Dynamic Trapezoidal Self Scheduling* (DyTSS). This algorithm is specially devoted to MC applications, and employs the aforementioned characterizations to split the samples to simulate among the available resources. Unlike previous self-scheduling algorithms, which decide the task distribution and the *chunk* size at the beginning of the execution, DyTSS creates every *chunk* at the moment of its execution, thus adapting the submission of jobs to any change in the number, performance or size of the resources being used in the dynamic environment.

As manually controlling all these tools would be difficult and tedious for the final user, a new framework called Montera (*MONTE Carlo RApido* – Fast Monte Carlo, from its Spanish acronym) is presented to overcome this issue. Montera carries out all the steps involved on a grid execution of MC codes, from information retrieval and *site* characterization to advanced scheduling via DyTSS algorithm.

These tools will be thoroughly explained together with the most significant details of the implementation. The rest of the paper is organized as follows: Section 2 is devoted to show the related work on the area; Section 3 explains the creation of Montera from a theoretical point of view and details the most significant characteristics of its implementation; in Section 4 the performance an the induced overhead are analysed, showing the behavior of Montera on real production infrastructures; Section 5 shows the conclusions and lessons learned during the development of this work.

2 RELATED WORKS

Job scheduling on grid infrastructures is a widely studied field, and a complete description of the research in this area falls out of the scope of this work. If a general reference is desired, Dong and Akl [17] have performed an impressive analysis of the state-of-the-art scheduling algorithms for grid computing, and a study on classical self-scheduling algorithms can be found in Chronopoulos' work [11] However, these approaches to the problem, although positive, do not offer a solution or a significant improvement to the execution of MC codes, as will be demonstrated in the following analysis. This achievable and necessary improvement is the aim and justification of the current work.

Note that the final objective is to reduce the *makespan* of a CPU-intensive application. Thus, techniques like those detailed in Li's work [32], which focus on

QoS or Yu's work [57] which focuses on data intensive applications, are beyond the scope of this analysis.

It is important to regard that the definition of grid computing is far away from being achieved among the scientific community. In this case the definition established by Ian Foster [3] is followed. Here, a three-point checklist is proposed to determine whether a given system can be considered a grid infrastructure or not. Regarding this list, it is a system that:

- coordinates resources that are not subject to centralized control
- uses standard, open, general-purpose protocols and interfaces
- deliver nontrivial qualities of service.

The first point means that the resources can belong to different organizations, each one with different software, usage and security policies. This resource sharing must be highly controlled, so users, service providers and infrastructure administrators are aware of *when*, *to whom*, *for what* and *under which conditions* a certain resource is shared.

By requiring that the protocols and interfaces (*middleware*) are general-purpose the resulting infrastructure is not oriented to solve a certain kind of problem and/or application, but can be employed to different areas of knowledge. The employment of standard, open protocols encourages the users to adapt the tools to their specific needs, while avoiding fragmentation.

In addition, the resulting infrastructure must be reliable and stable enough to constitute a valid alternative to the users, so they can carry on their experiments.

Of course, the existence of Montera as an specific tool to execute MC codes does not collide with the second point of the list: that definition is devoted to the middleware, and allows building more sophisticated layers on top of it.

2.1 Task Replication

Given that each simulation by an MC code is fully independent, and all of them are equally valid for obtaining the final result, task replication represents a powerful tool for reducing execution time. With this approach, the code starts the execution of more tasks than would be strictly necessary. Then, when the desired number of simulations has been executed, the remaining tasks are aborted. With this approach, it is possible to avoid bottlenecks resulting from performance loss or the failure of a particular resource, thus minimizing *makespan*.

This technique has been previously studied within the framework of grid computing [33, 38, 48, 53]. The results provided by the authors of each approach are highly variable, and they depend on the size of the problem, the number of tasks to execute and the status of the grid infrastructure employed to make the measurements. It is important to bear in mind that Poletti's work [53] is the only one performed in a real grid infrastructure, whereas the rest only provide the results of a simulation.

2.2 Task Grouping

As previously mentioned, MC codes are constituted by fixed sections executed at the beginning and end of the execution and the simulation of an arbitrary number of samples. If the aforementioned grid overheads are taken into account, and given that they are independent of the problem size, it makes sense to perform more than one simulation on each instance of the application.

Task grouping is a technique that has been widely applied in grid scheduling [56, 19, 55, 28, 16, 22]. Here, several tasks are grouped into a single job called a *chunk*. This technique minimizes some of the overheads produced by the grid execution model. The executable task is sent to the remote *site* only once, thus reducing transmission time, and the queue time has to be waited on only once per *chunk* instead of once per task. This approach is an adaptation of a technique for loop scheduling in distributed heterogeneous environments called "loop self-scheduling". Task grouping can be directly extrapolated to MC codes, considering that the *chunk* size corresponds to the number of samples executed in one task.

Nevertheless, in most of the current bibliography, key aspects related to a real dynamic grid production infrastructure – dynamicity of computing resources, performance variations and high fault rate – have not been taken into account or the performed tests have been made in simulated, controlled environments.

3 MONTERA

In this section, the components and functionalities of Montera, the proposed framework for the efficient execution of MC codes on grid infrastructures, will be described.

3.1 Characterization of Monte Carlo Codes

To improve the execution of MC codes on the grid, the first step of this work is to characterize them. The existence of a valid model allows the encapsulation of the application to be executed and its isolation from Montera, which can then seamlessly execute different codes.

Monte Carlo method obtains its name from the Principality of Monaco, "the gambling capital" [40]. It is based on the fact that the behavior of complex phenomena can be modeled by performing multiple independent, simple simulations, employing random or pseudo-random numbers to compute partial results and statistics to join them and obtain the problem solution [18].

MC applications – this is, applications based on MC method – are based on the simulation of an arbitrary number of independent samples, grouping them in one or more tasks. A typical MC code is divided into three different sections. The first section performs the common operations at the beginning of the execution, such as checking input data and initializing data structures. The second section is the simulation of the desired number of samples. The third section joins the results of the previous simulations and analyzes them to obtain the final result. The execution time of each section depends on the number of samples to simulate, and the time of the first and last may also incorporate a constant time. Thus, the execution time of the entire code depends on the number of samples plus a constant time.

In this proposal, the MC codes are represented by two parameters: *constant_effort* and *sample_effort*. *constant_effort* represents the effort necessary to execute the constant part of the application, and *sample_effort* is the effort necessary to simulate a single sample.

MC codes are CPU-intensive [9, 30] and in many cases are devoted to floating-point operations. Therefore, the chosen unit for the proposed parameters is *seconds· whetstone*.

The whetstone benchmark [12] is widely employed to estimate the speed of a computational resource when working with floating points. The unit in which the result is described is *Millions of Whetstone Instructions per Second* (MWIPS), which, for the sake of simplicity, will hereafter be shortened to whetstones. The whetstones of a *site* and the two mentioned parameters provide enough information to accurately estimate the execution time of a given instance of the code on a particular resource.

To obtain these values, a *code profiling* mechanism has been implemented in Montera. This mechanism submits to one or several *sites* a script that:

- executes the whetstone benchmark in the remote *site* to measure its performance
- executes the application to be profiled with an increasing number of samples, so *constant_effort* and *sample_effort* can be determined. This execution can last for a predetermined period or for the time the user considers necessary.

By repeating this process at different *sites*, the influence of exogenous factors, such as performance variations or benchmark imprecision, is minimized.

Finally, an analysis of the results based on the following equation is carried out:

$$Execution\ Time\ N\ Samples = Ce + N \cdot Se \tag{1}$$

where C_e corresponds to *constant_effort*, Se to *sample_effort* and N is the number of simulated samples in a given execution. As can be seen, obtaining C_e and S_e after two executions of the code with different number of samples is straightforward. However, to improve the accuracy of the results, this analysis is performed several times with an increasing value of N, and the parameters are obtained with a weighted average of the partial results. Finally, these values are normalized with the result of the benchmark execution.

3.2 Characterization of Grid Sites

When dealing with task scheduling on grid infrastructures, there are – as explained in Section 2 – two different approaches. The grid infrastructure can be considered either a black box or a set of known resources. Here, the latter is employed, so a strong effort has been put toward gathering as much information as possible about the composition of the infrastructure.

To accomplish this task, the *sites* have been characterized according to the following parameters:

- whetstones: efficiency when working with floating points
- queue time: average queue time for a task before being executed
- bandwidth
- number of successful and failed tasks in past executions
- available slots in past executions
- number of failed attempts to profile the *site* and when the last one occurred.

In the case of the first two parameters, both the average value and the typical deviations are stored.

With these parameters, Montera relies on more information about each *site* than previous tools, mainly based on its public information (e.g., architecture, CPU MHz). This approach represents a clear advantage for the task scheduling process, as a deeper knowledge of the grid infrastructure leads to more accurate decisions.

Montera is able to profile the *sites* of the grid infrastructure and to automatically obtain this information. When a new *site* is detected or a known one has been modified (e.g., different CPU or memory), Montera is capable of benchmarking it to obtain the previously detailed parameters. Although the employment of benchmarks to profile the *sites* and distribute tasks is not new [30], the incorporation of a tool to dynamically perform this action represents an improvement over previous works.

This benchmarking is fundamental to understand the performance of the infrastructure. Although the first time Montera is executed it presents significant overhead, the information is stored and recovered when needed, so it does not signify a drawback in a long-term approach. Montera also updates the information about the resources after each execution, thus increasing the knowledge about the infrastructure and improving the application performance in real time, all with a negligible computational cost. The bandwidth and whetstones are calculated based on the knowledge about the application, such as the size, model (*constant_effort* and *sample_effort*) and number of samples simulated, and the transmission and execution time of each *chunk*.

The result of the benchmark is also employed to control the availability of the *site*. After three failed benchmarking attempts, the *site* is removed for a fixed period of time from the list of resources to employ. Note that a benchmarking attempt is considered as failed if the expected result – an XML file containing whetstone results and several timestamps – is not provided, thus being able to detect both hardware and software failures. This way, Montera ensures that every task will be submitted to a valid resource, thus maximizing the probability of a successful execution.

The number of slots that were available in past is read at the initiation of Montera to decide the number and size of each task. After it has started and there are *chunks* running in different *sites*, this value is updated in real time. This approach is based on the idea that the number of slots will not vary much with time, so it is a good way of initiating Montera with no overhead and with acceptable precision.

The combination of the proposed characterization and Globus MDS (Monitoring and Discovery System) [47] – employed by GridWay and read by Montera – to gather information about the infrastructure resources is simple yet powerful. Compared with standard monitoring tools such as Nagios [2] or Ganglia [1] it is obviously much simpler and has little capabilities, but fits the purpose it was designed for on a perfect manner. The employment of whetstone benchmark leads to a precise estimation of the resource's capabilities, and storing the resources available on a given moment overcomes the problem of getting trustful information from a user's point of view: for example, knowing that a certain *site* has one thousand CPUs can lead to errors if it has a limit of twenty processes per user, and the existence of hyper-threading on a CPU will result in proving less performance than expected based on its chacrateristics. The proposed approach has also the advantage of not being invasive – not needing to install any software on the remote resource – and distributed, not employing a centralized resource to gather information.

It is noteworthy that Biessel's proposal [7] has been employed to calculate the average and typical deviation of an increasing number of values without the need to store all of them. Given the average and typical deviation of a set of N elements and a new element n, the authors propose an algorithm to obtain the average and typical deviation of the set composed by the $N + 1$ elements. This way, there is no need to store the entire set of values, and less computational resources are needed to calculate these parameters. It is also remarkable that this approach obtains the mean and typical deviation on a constant time, while traditional formulae scale linearly with the number of elements. Thus, Biessel's proposal represents both performance and storage gains, fully justifying its inclusion in Montera.

3.3 Characterization of the Grid Infrastructure

It has been demonstrated that the performance of a grid infrastructure can be modeled with only two parameters: *asymptotic performance* and *half performance length* [25, 41]. For the sake of completion, this model will be briefly described here.

From a computational point of view, a grid infrastructure can be seen as a collection of heterogeneous processors. Therefore, the number of tasks completed in a given moment can be described as

$$n(t) = \sum_{i \in G} N_i \lfloor \frac{t}{T_i} \rfloor \tag{2}$$

where N_i is the number of processors in the grid infrastructure G that can compute a given task in T_i seconds.

The average behavior of the system can then be represented as the evolution of $n(t)$. If done, a graph similar to Figure 1 [41] is obtained. Thus, a first-order description of a grid infrastructure can be made by

$$n(t) = mt + b = r_\infty t - n_{1/2} \tag{3}$$

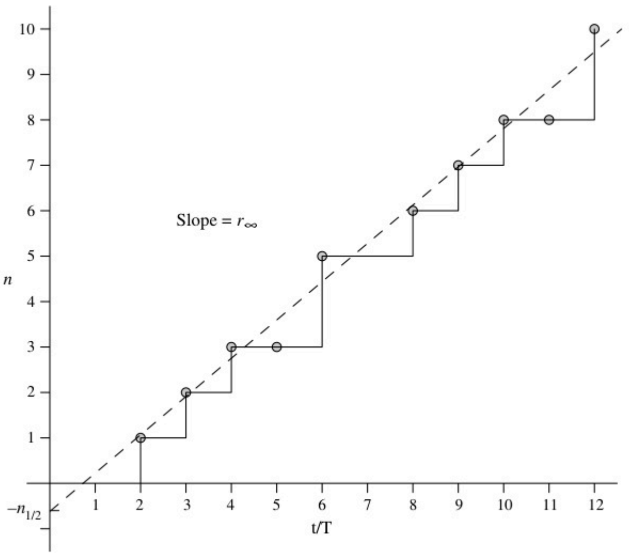

Fig. 1. Number of finished tasks as a function of time on an heterogeneous grid infrastructure

With r_∞ and $n_{1/2}$ being the *asymptotic performance* and half performance length as defined in [25]:

- *Asymptotic performance* r_∞: maximum rate of performance in tasks executed per second.
- *Half-performance length* $n_{1/2}$: number of tasks required to obtain the half of the *asymptotic performance.*

The problem with this approach is that it is based on the execution time of constant size tasks, and this work is focused on varying the task size. Thus, modification was necessary to represent the grid infrastructure behavior when executing MC codes. This adaptation is proposed here, together with an algorithm to calculate these parameters.

As explained in Section 3.1, MC codes have been modeled according to two parameters, *constant_effort* and *sample_effort.* Depending on the number of samples to simulate, the total computational effort needed to execute the task will vary. Also, the transmission time of the input/output data and queue time must be evaluated, because they represent a different overhead depending on the *chunk* size. All this information must be taken into account when calculating the performance.

Equations (2) and (3) show that it is necessary to know the execution time of each task to calculate the infrastructure performance, and "tasks per second" is employed as measuring unit. The problem with this approach is that Montera is focused on varying the task size, so this cannot be considered static and varies in execution time. Instead, it is now measured in samples/second simulated on the infrastructure. This unit embraces all the aforementioned overheads and accurately models the infrastructure behavior.

This new proposal makes the calculation of the performance significantly more complex. With the Montera approach, it is necessary to know the task distribution in order to calculate the infrastructure performance. However, when employing the DyTSS scheduling algorithm – which will be described in Section 3.4.3 – it is necessary to have information about the infrastructure performance to divide the simulation into different *chunks*. To avoid this deadlock, it is necessary to follow Algorithm 1.

Algorithm 1 Description of *create_task_list*() function

```
Require: $samples > 0$
  $siteStatus = read\_sites\_status()$
  $taskLlist = create\_simple\_task\_list(siteStatus)$
  $gridPerformance = calculate\_performance(taskList)$
  for $i = 0$ to 100 do
    $oldTaskList = taskList$
    $taskList = DTSS\_selfscheduling\_algorithm($
      $gridPerformance, siteStatus)$
    $gridPerformance =$
      $calculate\_performance(taskList)$
    if $taskList = oldTaskList$ then
      $Break$
    end if
  end for
  return $taskList$
```

This algorithm first reads the *site* status (obtained from past executions and the current status obtained with GridWay), calculates a naive task distribution and the infrastructure performance, this is, the ratio of simulated tasks per second.

After that, DyTSS algorithm is employed to calculate a better task distribution. Then again, this task distribution is employed to calculate the *site* performance. This loop is repeated until an optimum situation without changes between iterations is reached.

3.4 Scheduling Algorithms with Montera

In this work, a new scheduling algorithm called DyTSS (*Dynamic Trapezoid Self-Scheduling*) is proposed. DyTSS is based on the GTSS (*Grid Trapezoid Self-Scheduling*) algorithm and is adapted to focus on the execution of MC codes in extremely dynamic infrastructures.

3.4.1 GTSS Algorithm

The so-called *Grid Trapezoid Self-Scheduler* [22] algorithm is a modification of the *Trapezoid Self-Scheduler* (TSS) algorithm [51], being adapted to a grid environment. GTSS is based on three main elements: a grid benchmark model; the relationship factor $R_i^{\min}$; and the TSS dynamic algorithm.

GTSS employs the aforementioned grid benchmark model, using r_∞ and $n_{1/2}$ to adjust the *chunk* size. The heterogeneity factor, $R_i^{\min}$, is a coefficient that relates the execution times of every node in a grid infrastructure. This factor is defined as follows:

$$R_i^{\min} = \frac{\overline{T}_{\min}}{\overline{T}_{wall}(i)} \tag{4}$$

with $\overline{T}_{\min}$ being the minimum of all the execution times.

The behavior of GTSS for a given node j is then defined:

$$C_j^0 = \lceil F * R_{\min}^j \rceil \quad \text{being} \quad F = I/4n_{1/2}, L = 1 \tag{5}$$

$$C_j^i = C_j^{i-1} - D \quad \text{being} \quad D = \lfloor (F - L)/(N - 1) \rfloor, N = \lceil 2I/(F + L) \rceil. \tag{6}$$

Here, F is the size of the first *chunk*, L is the size of the last *chunk*, N is the number of steps, I is the number of tasks to be executed and is the resulting *chunk* size for each *site* j on the i^{th} iteration. F and I can be set statically or can employ the values proposed in Equation (5).

Beyond the mathematical definition, the idea that lies beyond this algorithm is to employ a decreasing number of tasks, balanced depending on the capabilities of each *site*.

The coefficient $R_i^{\min}$ represents the relationship between the execution time on the fastest resource and the resource i. Thus, employing it on Equation (5) ensures that the workload of each *site* will be directly related to its performance.

The existence of parameter D to reduce the number of *chunks* along the executions is proven to reduce *makespan*. As has been demonstrated [22], trapezoidal distributions outperform constant ones when dealing with heterogeneous systems.

3.4.2 Problems with Pure Self-Scheduling Algorithms in the Grid

During the development of this work, the execution of self-scheduling algorithms on the grid was intensively studied. In this analysis, a number of significant problems arose.

First, with this type of algorithm, the division of the tasks on different *chunks* is performed at the beginning of the execution. Although this approach is correct in static environments, it is unfeasible in grid infrastructures, which are, by definition, dynamic [3]. The status of every *site* is regarded as constant along the execution of the simulation, thus obviating an important characteristic of the grid. Moreover, the relationship between *sites* and *chunks* is not dynamic, so the failure or outage of a single resource results in an unfinished simulation. It can then be concluded that although self-scheduling algorithms perform extremely well in controlled environments, this lack of adaptability makes them sub-optimal in production infrastructures.

To overcome these problems, a new scheduling algorithm, DyTSS, has been created. The technique and parameters employed to divide the simulation into *chunks* is similar to that of GTSS. However, this division is not performed at the beginning of the execution, but every time a new *chunk* is desired. In this way, DyTSS is able to overcome the aforementioned issues, whereas at the same time profiting from the irregular *chunk* distribution of traditional self-scheduling algorithms to make the most of heterogeneous resources.

3.4.3 DyTSS Algorithm

As a particularity of this scheduling approach, the stages of grid characterization and scheduling must be performed simultaneously because of the strong relationship between them. Algorithm 2 shows a high level description of the necessary steps.

Algorithm 2 Description of DyTSS scheduling algorithm

```
Require: $desiredSamples > 0$
  $remainingSampeles = desiredSamples$
  $taskList = create\_task\_list(remainingSamples)$
  $submit(taskList)$
  while $remainingSamples > 0$ do
    for $task$ in $taskList$ do
      if $task$ finished $execution$ then
        $remainingSamples = remainingSamples - task.get\_number\_of\_samples()$
        $auxTaskList = create\_task\_list(remainingSamples)$
        $newTask = auxTaskList.pinck\_first\_task(task.getResource())$
        $submit(newTask)$
        $lastList.add(newTask)$
      end if
    end for
  end while
```

First, the information regarding the infrastructure is obtained, as has been already described. With this information, the simulation is divided among all the resources, assigning the same number of samples to each one.

Then, a loop is employed to adjust this initial distribution as follows:

- Calculate infrastructure performance with the current task distribution.
- Apply the GTSS scheduling policy to re-distribute the workload for each resource.

The complexity of this loop is linear on the number of resources. Anyway, given that it is only composed by several basic arithmetic operations per resource, its execution time remains negligible with the size of current grid infrastructures. Also, a limit of 100 iterations has been established based on the results of several experiments performed during the development of the algorithm: in the case of livelocks, they usually start before the tenth iteration, so the limit was set on the next order of magnitude. Even a finer tune could be performed, the authors consider that the cost/benefit proportion was favorable to a high limit, ensuring that the loop was not stopped before reaching an equilibrium or livelock.

Note that in this algorithm GTSS is always applied as for C_0^i in Equation (5). The task size decreases because I is reduced any time a previous task has ended, so the steps described in Equation (6) are not necessary.

There is still another innovative approach in the proposed scheduling process. The application of the scheduling algorithm is not employed to obtain the distribution of all the simulations to be performed, but only for a single task for each available slot. Then, when any task ends or a new resource is discovered, the algorithm is applied again, employing real-time information about the *sites* to obtain the size of the next task to be submitted. In this way, the task size is always calculated with the latest information, thus improving the effectiveness of the algorithm. Also, the failure or performance slowdown of any resource does not represent a bottleneck, as happens with the traditional self-scheduling algorithms.

Finally, a technical decision must be noted. The number of tasks submitted to any *site* is not the number of available slots but a 20 % greater. This increase is due to two complementary factors.

First, there is a reduction in the execution walltime. If a task is submitted only after the previous one has finished, an overhead is produced due to the result copied back to the local *site*, the current status checked and so. However, if the second task is already queued on the remote *site*, it starts its execution just after the first one ends, thus reducing *makespan*.

It is important to note that task grouping techniques (as described in the previous section) could be used instead of the dynamic creation of jobs in the execution time. The drawback is that, although this approach would reduce the aforementioned overheads, it would represent a flexibility loss. With this approach, the size of the task is dynamically determined by the status and performance of the entire grid infrastructure, which could not be done if all the tasks submitted to a single *site* were created at the same time and submitted all together.

The second reason is that the number of available slots in a given *site* can vary during the execution of the application. With this scheduling decision it is ensured that an optimal number of tasks will be created for each resource, while keeping the number of replicas low: if the number of slots is reduced, Montera will detect it and will reduce the number of submitted tasks, cancelling those still waiting so that no CPU time is lost; if it is increased, some of the 20 % spare tasks will start their executions, and Montera will detect it and will create more tasks until reaching the 20 % excess again. This functionality will be further analyzed in the Results section.

Note that the difference between the proposed scheduling and pure replication algorithms is subtle, but highly relevant. In both approaches, the number of tasks submitted is greater than what is needed, but their behavior is different. In the case of replication, the spare tasks are cancelled, thus losing the computational effort put into their execution. However, in the case of DyTSS – given that in MC codes all results are equally valid – all the results are employed in the simulation, and the only non-useful tasks are those being executed when the desired number of samples is reached. In this way, DyTSS obtains the advantages of replication, such as higher throughput and avoidance of bottlenecks, without suffering the overhead and abuse of the infrastructure inherent to replication algorithms.

3.4.4 Two-Level Scheduling

As described in Section 2, previous schedulers perform the task scheduling locally and decide the size of the *chunk* to submit to each remote resource prior to the proper execution. In Montera, a new approach is followed, allowing the performance of this scheduling on both the local and remote resources. The objective is to allow the *chunk* to dynamically adjust its size depending on the status of the remote resource and the user needs. To do this, Montera not only submits the desired task, but also includes all the available information about the *site*, the application to be executed, and several scripts in charge of adapting the *chunk* size, depending on the user requirements.

3.5 Montera Architecture

Figure 2 shows a high-level description of the Montera architecture. As can be seen, the architecture is divided into two different sections: Local Montera and Remote Montera. Local Montera is executed in the local resource, and Remote Montera is executed in the remote resources.

3.5.1 Local Montera

Local Montera is the core of the developed framework. Its purpose is to receive the job specifications and requirements from the user and execute them as efficiently as possible.

To obtain the information about the available grid resources, the Information Manager employs two tools: GridWay [26] and Remote Montera. GridWay is em-

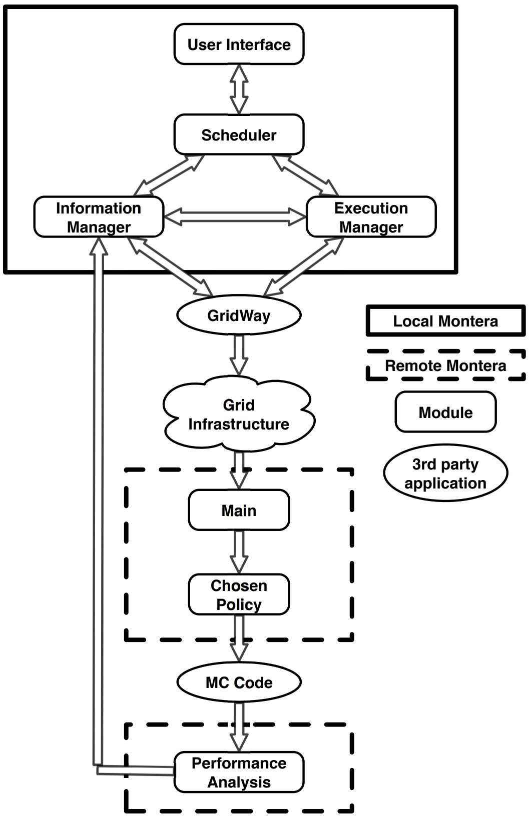

Fig. 2. High level description of Montera architecture

ployed to gather the static information the *sites* publish about themselves and the number of tasks being executed at a given moment. Remote Montera provides information about the efficiency of the *sites* after the execution of each task as well as the results of the benchmark executed – if needed – to calculate their real performance.

With this information, the Scheduler decides the ideal size and number of *chunks* to execute on each *site*. Finally, the Execution Manager carries out these executions. It employs DRMAA [50] to perform the job submission and control.

Distributed Resource Management Application API, DRMAA [50] is a high-level API specification for the submission and control of jobs to one or more *sites* within a grid infrastructure. It defines operations covering all the steps of a grid execution: submission, monitoring and retrieval of the jobs.

The usage of DRMAA to implement our application allows controlling all the steps involved on the execution of tasks in an unattended way: automatically creating an arbitrary number of tasks, executing them remotely and bringing the results back. To carry this out, only a metascheduling implementing DRMAA API is needed, allowing Montera to delegate these tasks on GridWay.

Montera allows the employment of different scheduling policies. To do this, the Scheduler defines an interface that covers the steps of *chunk* creation and control. Thus, defining a new policy only requires implementing the interface with the desired functionality.

At this point, it is important to point out that Montera does not strictly depend on GridWay metascheduler but could rely on any other metascheduler implementing DRMAA with slight modifications to the code. Nevertheless, GridWay provides an interesting set of tools that makes it particularly suitable for this work. Also, studies like Poletti's [52] suggest that, in addition to GridWays' capabilities for job management and control, its performance is clearly superior to other widely used tools such as WMS.

An important architectural decision when designing Montera was determining which of the two grid submission paradigms was more convenient, schedulers or pilot jobs. The problem with pilot jobs is that they currently run only on systems where Network Address Translation (NAT) capabilities are available, preventing the use of software in infrastructures not based on the grid middleware developed inside the EGEE gLite projects. In addition, since pilot-jobs systems basically act as wrappers for retrieving tasks, they could be also implemented in other way within any scheduler in order to improve the performance efficiency and, at the same time, profit from the migration possibilities, i.e. when adapting Montera to any scheduler it is possible to profit from the pilot-jobs advantages in a future.

3.5.2 Remote Montera

The second part of the proposed framework, Remote Montera, optimizes the execution of the application on the remote *sites*. Remote Montera is copied to the *site* together with the application to execute, and it decides the exact number of samples that will be simulated in that *chunk*.

Due to the heterogeneity of the remote *sites* and the available tools for each, Remote Montera has been implemented as a set of shell scripts to make it as portable as possible. Its size is greatly reduced, so the overhead induced by its copy to each resource is negligible.

The first script, Main Script, reads the input data, user requirements and information stored regarding the performance of the *site*. Then, the second script, Chosen Policy, calculates the exact number of samples to be simulated and performs the execution. Finally, the Performance Analysis is carried out, and the results are returned to Local Montera.

Montera includes a *Chosen Policy* set to decide the size of each task on each *site*. When the user submits a job to Montera, the user chooses which one to employ, and it will be submitted to the remote *site*. Depending on the desired functionality, the behavior of this script can be completely different. For example, *Deadline* calculates how many samples to execute before a given moment regarding the job submission date, the queue time and the bandwidth. In *Flexible*, Local Montera provides a maximum and minimum number of samples to simulate, and the remote script decides the exact number regarding the *site* performance and its current status. Note that the possibility of combining different local and remote policies allows the application of widely differing scheduling algorithms, so the user can adapt it to the size and characteristics of the experiment to be simulated.

As in Local Montera, Remote Montera also allows the user's own scheduler to be employed instead of the default schedulers. Creating a new scheduler only requires programming a shell script with the desired behavior. Montera provides information that can be used for this purpose (e.g., submission time, status of the resource) and, if something else is required, it can be provided via the job template.

3.5.3 User Interface

During the development of Montera, much effort was invested in creating an easy-to-use application while at the same time providing the user the necessary tools to implement different policies.

Any scheduling policy must implement an interface with just two methods: `create_chunk_list` and `control_execution`. First, the user defines the number and size of *chunks* to simulate in the first place. Then, the user specifies how the execution will be controlled. The user can check whether the execution of any *chunk* was successful, can create and submit new ones or can cancel the desired ones.

A typical Montera execution is determined by the input file being employed. This file indicates the application to execute, number of samples to simulate, input and output files and the chosen scheduling policy. This is probably the most simple command line-based interface possible, and ensures that any user will be able to perform the desired simulations in a very short amount of time.

3.5.4 Integration of a New Application

Given that Montera is designed to execute different applications, the integration of a new one is expected to be seamlessly performed.

Montera expects that the application has a determinate number of parameters, and that they are always located in the same position:

```
<application name> <number of tasks to simulate> <input parameters>
```

If this would not happen the user had to write a wrapper, a task that can be seamlessly performed with a simple shell script.

3.6 Limitations of the Montera Approach

As stated in the Introduction section, it is important to bear in mind that the proposed code characterization and scheduling algorithm can only be applied to a subset of Monte Carlo codes, the stateless ones. On stateful Monte Carlo codes the problem cannot always be divided on independent executions of arbitrary size, as the result of a given simulation depends on the previous ones. Random Walks, for example, employ random numbers to determine the route of a particle through a user-defined space, and the result of each sample – i.e. one step – depends on its position prior to the advance.

This restriction makes the study of some areas of knowledge beyond the scope of Montera: on protein folding problems, algorithms based on Monte Carlo replica sampling [54], Metropolis Monte Carlo [39] or random walks [42] are employed [44, 13, 58, 6]; lattice QCD [31] employs random walks to determine the position of particles in the space-time; in lattice simulations, Monte Carlo is used for a generation of a probability distribution [35]; in computational chemistry, Metropolis Monte Carlo is widely employed to generate different configurations of the system [29], together with more advanced algorithms [20].

Anyway, performing a deep analysis on the usage of different Monte Carlo-based algorithms on scientific applications is beyond the scope of this work. If more information is desired, the proposed references provide a gentle introduction on the area.

4 RESULTS

Here, the results of the Montera scalability and performance analysis will be detailed. To conduct the analysis, a grid simulator was built to compare the theoretical and practical aspects of the work. Then, several experiments were performed with different scheduling policies and application requirements to demonstrate the feasibility of the proposed approach on a production grid infrastructure.

4.1 Testbed

The performance analysis of Montera was performed on two virtual organizations, namely *fusion* and `prod.vo.eu-eela.eu`. The fusion virtual organization, deployed by the EGI grid infrastructure (see `http://www.egi.eu`), counted on 25 *sites* that executed more than 325 000 jobs in 2010 only taking into account European *sites*, which was equivalent to 2 708 319 KSI2K (Kilo SpecInt2000 [43]) hours. Of course, not all of these CPUs can be employed by a single user at the same time because strict usage policies are implemented to prevent abuses. This is a real production environment with diversity of resources (cores from 1 GHz to 3.2, for example), the status of which can be monitored in real time at `http://www3.egee.cesga.es`. On the other side, GISELA (Grid Initiative for e-Science virtual communities in Europe and Latin America) project deployed an infrastructure oriented to the collaboration between Europe and Latin America research centres, the general purpose VO of which is `prod.vo.eu-eela.eu`. This VO is currently deployed by the Latin American and Caribbean Grid Initiative IGALC (see `http://www.igalc.org`), which was established also in 2010. Even when there are still not yearly usage statistics, preliminary results [14] show that around 500 000 KSI2K hours were accounted on the first 6 months of existence.

In order to perform the necessary experiments two different MC codes have been employed, namely FANFER2 [49] in its grid version [45] and ISDEP [10]. They have been chosen because of their deep differences on their behavior. FAFNER2 simulates many particles on a fast manner, employing several seconds on each one. On the other side, ISDEP needs about one hour on a standard PC to simulate a single trajectory. This way they represent both extremes. If Montera is able to speed both up, it will be proven to be effective on a wide range of applications.

FAFNER2 is a 3D code that simulates the neutral beam injection (NBI) technology on fusion devices. FAFNER2 models the injection of fast neutral particles into toroidal plasmas and the orbits of the resulting ions [34]. Thus, the global efficiency, the losses (i.e., shine-through, charge exchange and orbit losses, including those to limiters), the birth profile and the heating profile can be calculated for different discharges. Because of the design of the code, a single instance of FAFNER2 can only simulate up to 8 000 samples, which establishes an upper bound to the size of each *chunk*.

ISDEP (Integrator of Stochastic Differential Equations for Plasmas) [10] is a MC code that solves the plasma dynamics in fusion devices. It is based on the equivalence between the Fokker-Planck and the Langevin equations. Basically, it simulates the trajectory of fast ions inside the plasma. The MC nature of ISDEP makes it suitable for being executed by means of independent tasks. Therefore, it has been employed in several BOINC initiatives such as Zivis and its followed-up phase Ibercivis, in desktop computing projects such as EDGES and on grid projects too, for example the EGEE series or EUFORIA.

The local resource on which Montera was executed consisted of a virtual machine running Scientific Linux 4, with GridWay 5.4.0 and Java Virtual Machine 1.5.0.09. This resource was in production status -being employed by different users to perform their daily tasks with a quite restrictive configuration: a scheduling interval of 30 seconds, a dispatch *chunk* of 15 tasks per scheduling interval, and a maximum number of simultaneous jobs per user of 100.

These restrictions do, in fact, limit the scalability of the proposed solution, add overhead and hinder the obtainment of a greater degree of parallelism. The decision not to modify the configuration (i.e., tuning the GridWay configuration to obtain the best possible results) was part of the determination to create an application and scheduling algorithm that outperforms the existing tools in production environments. This particular configuration was established by the system administrators of the local resource as being the most convenient regarding the available hardware and computational demands. Also, this GridWay instance was being employed by several users while the experiment was being conducted. Thus, the possibility of adjusting the system was completely eliminated.

4.2 Simulation of the DyTSS Algorithm

Montera includes a simple yet effective grid simulator. Coupled with the rest of the code, it models the execution of the simulation on a grid.

This simulator uses the information about the application and the *sites*' performance obtained in past executions of the code, thus being able to accurately calculate the execution time as well as the expected queue time and how long the data transmission will take. The simulator also employs the number of available resources in the previous executions – which were obtained as explained in the previous section – to calculate the number and size of the *chunks*.

Although the simulator obviates some important characteristics of the nature of the grid (dynamism, heterogeneity and fault rate), it is still useful for providing an estimation of the performance of any scheduling and as first approach for the initial submission of jobs.

Figures 3 and 4 show the simulation of two different scheduling policies, namely, *EqualChunkSize* and DyTSS. It can be seen that in these simulations the employment of the DyTSS algorithm does not represent a significant improvement over a simple scheduling policy because of the partition of the samples to be simulated in *chunks* of equal size. This is due to the simplicity of the simulation, which was done in a controlled environment: there are no performance losses, new resources or any other issues, so the employed algorithm does not need to adapt the number or size of the *chunks*. As has been explained, one of the advantages of DyTSS over the rest of the scheduling algorithms is its robustness, which is not visible here.

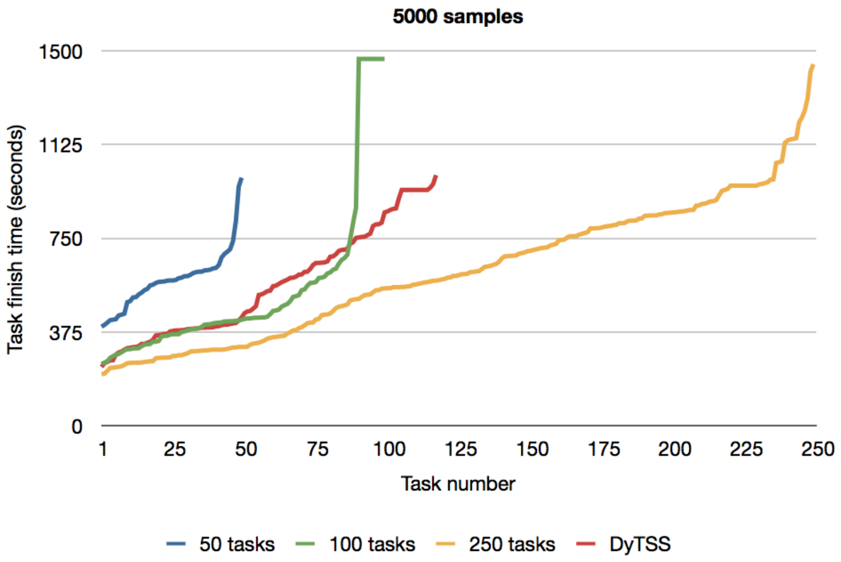

Fig. 3. Simulation of a 5 000-sample FAFNER2 execution divided into a different number of equally sized tasks and with the DyTSS algorithm

As can be seen, with the *EqualChunkSize* policy the results vary depending on the *chunk* size. This is due to the status of the grid, with about 90 available slots for a single user with the previously described GridWay configuration. In the first case, all 50 tasks start their execution immediately, so there is no queue overhead,

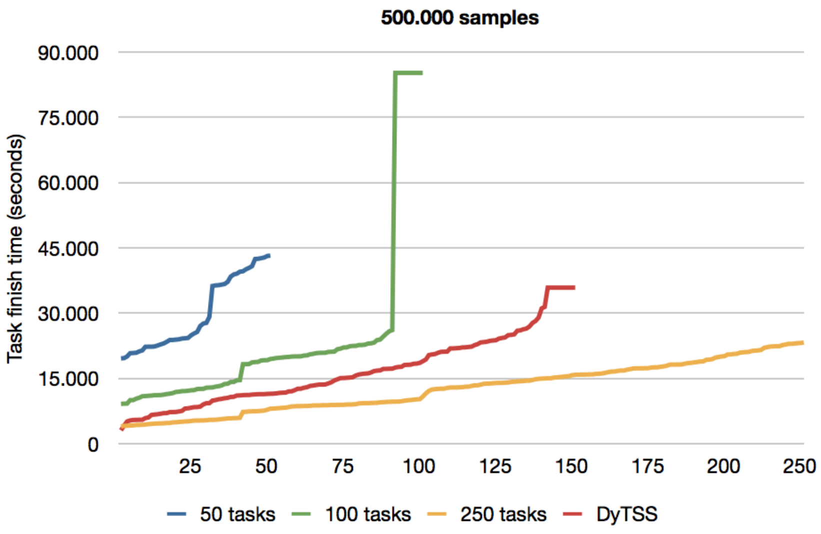

Fig. 4. Simulation of a 500 000-sample FAFNER2 execution. Conditions are the same as in Figure 3.

and the walltime corresponds to the slowest resource. In the 250-task case, there is a significant queue time in two-thirds of the tasks, but having a larger number allows for better scheduling, sending more tasks to the fastest processors. Finally, the 100-task division presents the inconveniences of the two others, but none of their advantages: there are 10 tasks that have a significant waiting time, but they are not enough to perform any kind of useful scheduling. This fact is depicted in the slope of the growing curves. A high or even infinite value indicates that the tasks are waiting for their execution, and low or even null value indicates an immediate execution of the tasks. DyTSS (Montera) grows with a constant value, so the tasks are progressively executed and, at the same time, the whole calculation lasts for the least possible amount of time.

In other words, in the case of DyTSS, the number and size of tasks has been adapted to the grid infrastructure, so the execution is always efficient. It is also noteworthy that the DyTSS result graph is nearly a straight line (constant slope) with no vertical steps, as with the others. These steps appear at the point where, for a long time, no tasks have finished, which usually indicates a bottleneck or performance loss.

4.3 Grid Executions

After performing the previous simulations of DyTSS in a controlled environment, different experiments were performed to check the feasibility of Montera in a real grid environment.

Two different experiments were performed: *Deadline Policy*, where all the different components of Montera were employed to simulate as many samples as possible before a given deadline; and *DyTSS Policy*, where the proposed algorithm is compared with others to demonstrate its superiority.

4.3.1 Deadline Policy

As has been explained, many different scheduling policies can be employed with Montera. In this case, a *Deadline Policy* is proposed, where Montera submits a task to each free slot and executes as many simulations as possible before a given deadline.

This is an interesting policy because it offers a new possibility that, to the authors' knowledge, was impossible to efficiently perform with the previous scheduling algorithms and tools. This policy is divided into two sections, one in Local Montera and another in Remote Montera.

In Local Montera, a new policy was created. This policy creates one task for each free slot and includes the submission time as an environment variable, which is available in the remote script. In Remote Montera, the remote script that determines the number of samples to simulate reads this environment variable, the local system date, the *site* performance and the application profiling, and it calculates the number of samples that could be simulated before the desired threshold. In this way, the queue time, task cancellation or any other grid-related issue will have no influence on the accomplishment of the deadline. If a task starts its execution, a certain number of samples will be immediately sampled, and if it stays in the queue for a long time, it will simulate less.

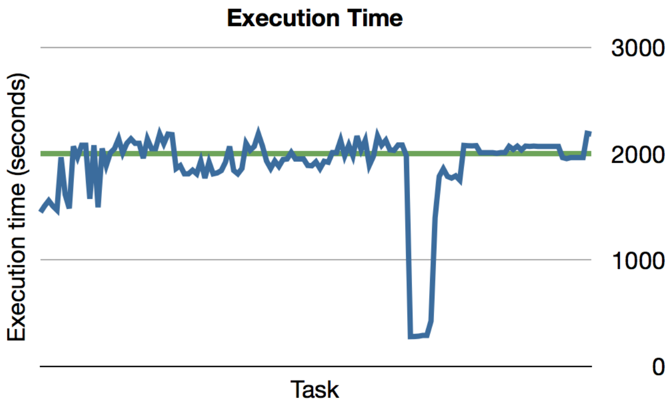

Fig. 5. Deadline policy. Task finalization is represented in blue, and the desired ending time (2 000 seconds) is in green

The results of this policy can be seen in Figure 5, which shows the ending time of every task after requiring Montera to simulate as many samples as possible before a given deadline (2 000 seconds). To provide realistic results, only an execution was performed. Thus, certain tasks failed and most of the others did not perfectly accomplish the deadline requirements. Aside from this small lack of precision – inherent to any grid execution – it is clear that the proposed characterization of MC codes and grid *sites* is accurate enough to be used for significant simulations on production infrastructures.

4.3.2 DyTSS Scheduling Policy

Here, the feasibility of the DyTSS scheduling algorithm is proved against two different policies, an *Equal Size* distribution and GTSS. These policies have been chosen because they are among the most representative ones: *Equal Size* is a basic policy that is widely employed for these kinds of problems; and GTSS has been demonstrated – as explained in Section 2 – to be the most efficient self-scheduling algorithm on controlled grid infrastructures.

In the case of the *Equal Size* policy tests, some adjustments were made to maximize their performance. A rescheduling threshold of 300 seconds and a maximum queue time of 600 seconds were established to avoid saturated resources and to start their execution as soon as possible. Also, the GridWay default scheduling policy – the efficiency of which has been demonstrated [27] – was employed to choose the *site* at which each task is executed. These optimizations were performed to ensure that any improvement obtained with Montera and DyTSS was only due to the application and its scheduling algorithm, not to exogenous factors such as relying on the capability of GridWay to submit and manage the remote execution of tasks.

Some statements about the GTSS tests must also be made. As noted in Section 3.4.2, the information about the infrastructure must be obtained prior to the execution of any self-scheduling algorithm. In this case, the information was obtained with Montera, employing the previously described tools. The alternative option of using public information about the *site* leads to incorrect results and errors. For example, in the local *site* at CIEMAT (`ce01-tic.ciemat.es`), 208 nodes have been published, but only 20 can be employed by a single user at the same time. If this information alone was employed, the number of submitted tasks would be extremely large, thus affecting the performance negatively.

With FAFNER2 three experiments with increasing task size were performed. The number of samples was 5 000, 25 000, 50 000, 200 000 and 400 000 (from 8 to 180 CPU hours), which can be considered as short, medium and long simulations. As can be seen from Figure 6, different policies lead to different results in terms of *makespan*, and some are more suitable than others, depending on the number of simulations to perform.

In the case of *Equal Size*, results are consistent with those obtained with the simulator (see Figures 3 and 4 for details). With a small number of particles, the grid execution tends to benefit from a small number of tasks because the overheads are significant and represent a high proportion of the total execution time. However, when the size of the problem grows, the overhead is not as important. An accurate scheduling is now of vital importance, and it is better performed with a greater number of tasks to distribute.

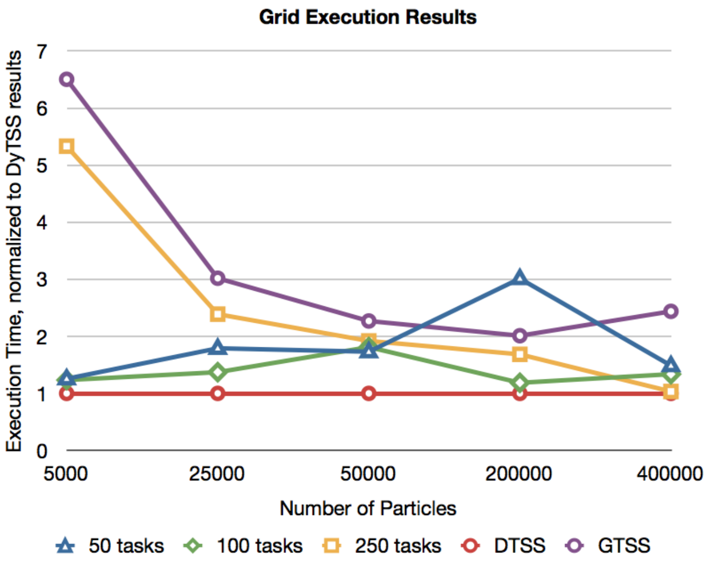

Fig. 6. Execution of FAFNER2 with different policies and an increasing number of particles. Results are normalized to DyTSS *makespan*.

The results obtained with GTSS were not as good in the real environment as in the simulations. As previously explained, GTSS creates a fixed relationship between tasks and resources. Thus, the failure or performance loss of a single *site* can slow down the entire simulation and increase *makespan*. Although the information obtained with Montera was employed to minimize these issues, the result of employing the algorithm on dynamic infrastructures proved to be suboptimal.

Finally, DyTSS obtained the best results in every single case. From over 650 % in the best case to 3 % in the worst one, its dynamic task distribution and adaptation to unstable resources resulted in the best alternative among those chosen.

In this experiment, the limit of 400 000 particles is forced by the application design. As it has been stated, FAFNER2 forces an upper limit of 8 000 particles per task, and the smallest distribution with Equal Size has 50 tasks, so the 400 000 particles limit arises. Anyway, a typical FAFNER2 experiment employs about 4 000 particles on a serial execution and 8 000 to 80 000 for parallel ones. Thus, demonstrating that DyTSS outperforms all the other algorithms with up to 400 000 particles can be considered significant.

In the case of ISDEP, a minimum and valuable physical research requires of fifty thousand trajectories (50 000). In this paper, results related to 5 000 trajectories are presented; such a number has been selected since it clearly represents the benefits for using ISDEP with Montera as a proof of concept without performing the whole research, the results and analysis of which are out of the scope of this paper. The way that such a whole physical research works, which is explained here for the readers' convenience, is that thousands of identical jobs are submitted only differing on a random seed.

The execution of ISDEP presents a particularity that has to be pointed out, also helping understand the behavior of Montera: depending on some input parameters related to the precision and size of the time steps, the execution time of ISDEP is highly variable, so making no distinction among the different executions would lead to erroneous profiling and scheduling. Luckily, Montera stores each application profiling employing a user-defined string name as key. Thus, determining a name like ISDEP_WITH_X_PRECISION to store the application would be enough to ensure that the different ISDEP instances are clustered into groups of similar behavior.

Table 2 shows the results of this execution. Likely to FAFNER2, algorithms GTSS and *Equal Size* with 250 and 500 tasks have been compared with DyTSS.

It is important to notice why in the case of ISDEP the Equal Size task distribution employed have been only those with 250 and 500 tasks. The experiment was designed to be identical as the one with FAFNER2, but a problem arose: most of the grid *sites* have a limitation on the task execution time, which usually varies between 24 and 48h. Given that each simulation on ISDEP takes about one CPU hour, tasks with more than 20 simulations have high probabilities of being cancelled (and even this phenomenon actually happened in the preliminary tests performed in this work). Thus, the chosen distributions have been 250 and 500 tasks, simulating 20 and 10 particles per task.

For the sake of completion, Table 2 also includes the execution time normalized to DyTSS (as in Figure 1) and the standard deviation.

Samples	*50 tasks (seconds)*	*100 tasks (seconds)*	*250 tasks (seconds)*	*GTSS (seconds)*	*DyTSS (seconds)*
5 000	1 331 ±85.56 %	1 307 ±55.77 %	5 649 ±65.96 %	6 892 ±49.11 %	1 059 ±48.91 %
25 000	5 553 ±58.79 %	5 304 ±62.06 %	5 568 ±53.77 %	8 038 ±50.03 %	2 301 ±43.34 %
50 000	5 857 ±61.37 %	6 131 ±44.06 %	6 496 ±9.88 %	7 681 ±15.45 %	3 385 ±23.82 %
200 000	15 665 ±12.12 %	6 587 ±9.07 %	10 701 ±29.42 %	10 456 ±5.56 %	5 200 ±11.57 %
400 000	16 647 ±15.17 %	14 963 ±25.03 %	11 568 ±10.63 %	27 284 ±21.98 %	11 192 ±12.03 %

Table 1. Execution time and standard deviation of the values shown in Figure 6. Deeper explanation through the text

Tables 1 and 2 show the 5-averaged execution time and standard deviation for all the experiments performed with FAFNER2 and ISDEP, respectively. As can be seen, DyTSS always offers the best execution times and usually the most precise results in terms of the execution time. The dynamicity of the grid infrastructures and variety of resources ensure that the execution time will hardly be the same among different executions, but this standard deviation can give a estimation of how the different algorithms work anyway. A reduced deviation means that the scheduling algorithm is able to overcome small differences on the infrastructure between different executions, and a high deviation means that the result are highly dependent on the specific site executing each task.

It is easy to see why the standard deviation is usually higher when the number of tasks on which the simulation has been splitted is smaller: the number of samples to simulate on each task is bigger and so is the computational effort, so the performance of the *site* executing that task becomes more important. Also, as has been previously described, a smaller number of tasks limits the scheduling possibilities and the obtention of a satisfactory execution time becomes a matter of luck.

parameter	*250 tasks*	*500 tasks*	*GTSS*	*DyTSS*
Execution time [s]	253 812	222 061	249 331	183 956
Makespan normalizad to DyTSS	1.35	1.21	1.33	1
Standard Deviation [%]	7.04	32.30	29.68	6.96

Table 2. Results of the simulation of 5 000 particles with ISDEP with different scheduling policies. Deeper explanation through the text.

There is still an additional advantage of employing Montera instead of a traditional scheduler. As all the decisions about task distribution and scheduling are managed by the application itself, the final user does not need to be aware or spend any time on performing these operations. This represents a significant improvement for final users, who can now focus on their research area and delegate non-essential issues to his specific piece of software.

4.3.3 Dynamic Adaptation to Free Slots with DyTSS Scheduling Policy

As already noted, with DyTSS Montera is able to detect the number of free slots in any resource and dynamically adapt the number of tasks to be executed. In Figure 7, a proof of this functionality in a production *site* is graphically detailed.

This experiment consisted on the execution of DyTSS on a single grid *site*, `ce01-tic.ciemat.es`. After the benchmarking of the *site*, the execution started with just one task. Montera submitted its 20 % more tasks, which was one more in this initial case because Montera adjusted this calculation (number of tasks by 1.2) to the lowest integer. As there are free slots, this second task starts its execution immediately. When Montera detects it, it creates another task to accomplish the 20 % margin again, and so on. This process is then repeated until 20 tasks are executed simultaneously, thus reaching the number of free slots that can be employed by a single user on this *site*. Afterwards, the tasks were only created and executed after the ending of the previous ones as no free slots were detected.

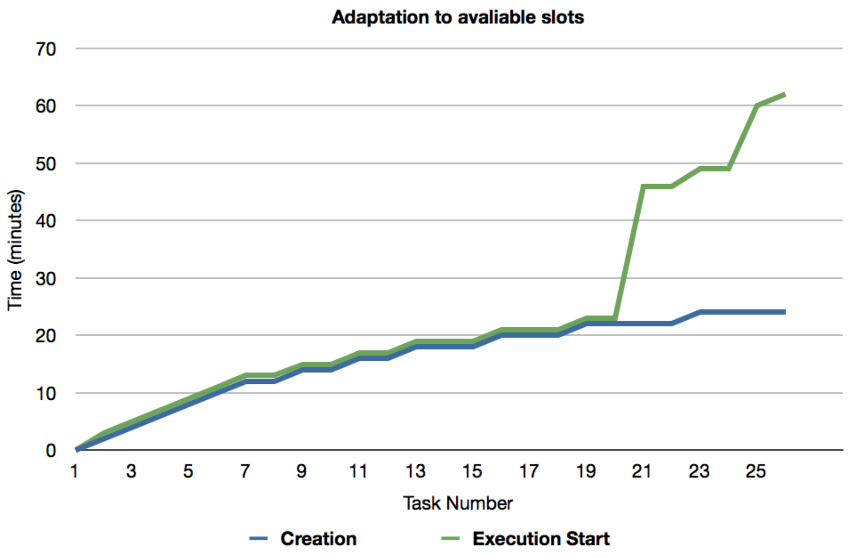

Fig. 7. Adaptation to the number of available slots on a *site* with 20 free slots

When this simulation was finished, the information about the number of free slots was stored. In the future, this information will be employed to determine the number of tasks to be created at the beginning of the execution.

5 CONCLUSIONS

In this work, the need for improvement in the efficiency of MC codes on the grid has been demonstrated.

After presenting the problem, a newly created algorithm called DyTSS is proposed to solve it. This algorithm has the advantage of being focused on a specific kind of application and infrastructure, thus being able to employ a wider set of tools and techniques than general purpose algorithms, such as characterizations of infrastructures, *sites* and applications.

A scheduling application called Montera was then presented. Montera implements mechanisms to collect the aforementioned characterizations, and it provides the user with the ability to perform a two-level scheduling. This approach allows the user to modify the execution depending on his/her needs and the requirements of the experiment.

As has been shown in the Results section, the executions of FAFNER2 with Montera clearly overcome the rest of the alternatives in terms of *makespan*. Thus, it is proven that the employment of Montera is justified for Monte Carlo-based grid executions.

With MC codes being widely employed in many different areas of knowledge, it is expected that a new tool that simplifies its grid execution while boosting its performance will be well received in the scientific community. Also, because it is being developed with real world applications in collaboration with users, its correct behavior and usability is guaranteed thanks to the direct feedback between the development team and the final users.

Acknowledgments

The authors would like to thank Mr. Antonio J. Rubio-Montero for the valuable discussions during the development of this work.

REFERENCES

[1] Ganglia Web: `http://ganglia.sourceforge.net`.

[2] Nagios Web: `http://www.nagios.org`.

[3] Foster, I.: What is the Grid? A Three Point Checklist. Grid Today, 1, 22/07/2002.

[4] Andreo, P.: Monte Carlo Techniques in Medical Radiation Physics. Physics in medicine and biology, 36:861, 1991.

[5] Arellano, M.—Bond, S.: Some Tests of Specification for Panel Data: Monte Carlo Evidence and an Application to Employment Equations. The Review of Economic Studies, Vol. 58, 1991, No. 2, pp. 277–297.

[6] Bastolla, U.—Frauenkron, H.—Gerstner, E.—Grassberger, P.—Nadler, W.: Testing a New Monte Carlo Algorithm for Protein Folding Proteins. Vol. 32, 1998, No. 1, pp. 52–66.

[7] Biesel, H.: Recursive Calculation of the Standard Deviation with Increased Accuracy. Chromatographia 1977.

[8] Boyle, P. P.: Options: A Monte Carlo Approach. Journal of Financial Economics, Vol. 4, 1977, No. 3, pp. 323–338.

[9] Brooks, S.: Markov Chain Monte Carlo Method and Its Application. Journal of the Royal Statistical Society: Series D (The Statistician), Vol. 47, 1998, No. 1, pp. 69–100.

[10] Castejón, F. et al.: Ion Kinetic Transport in the Presence of Collisions and Electric Field in TJ-II ECRH Plasmas. Plasma Physics Controlled Fusion, Vol. 49, 2007, pp. 753–776.

[11] Chronopoulos, A. T.—Benche, M.—Grosu, D.—Andonie, R.: A Class of Loop Self-Scheduling for Heterogeneous Clusters. Proceedings of the 3rd IEEE International Conference on Cluster Computing 2001, pp. 282–291.

[12] Curnow, H. J.—Wichmann, B. A.: A Syntetic Benchmark. Computer Journal, Vol. 19, 1976, No. 1, pp. 43–49.

[13] Czibula, C.—Bocicor, M.—Czibula, I.-G.: Solving the Protein Folding Problem Using a Distributed Q-Learning Approach. International Journal of Computers, Vol. 5, 2011, No. 3, pp. 404–413.

[14] Diacovo, R.: Gisela Infrastructure Status Report. EU Deliverable: D4.1. Universidade Federal do Rio de Janeiro, pp. 1–14, 2010.

[15] Diaconis, P.: The Markov Chain Monte Carlo Revolution. American Mathematical Society, Vol. 46, 2009, pp. 179–205.

[16] Díaz, J.—Reyes, S.—Niño, A.—Muñoz-Caro, C.: Derivation of Self-Scheduling Algorithms for Heterogeneous Distributed Computer Systems: Application to Internet-Based Grids of Computers. Future Generation Computer Systems, Vol. 25, 2009, No. 6, pp. 617–626.

[17] Dong, F.—Akl, S. G.: Scheduling Algorithms for Grid Computing: State of the art and Open Problems. School of Computing 2006.

[18] Eckhard, R.: Stan Ulam, John Von Neumann and the Monte Carlo Method. Los Alamos Science, Special Issue 1987.

[19] Fann, Y.-W.—Yang, C.-T.—Tseng, S.-S.—Tsay, C.-J.: An Intelligent Parallel Loop Scheduling for Parallelizing Compilers. Journal of Information Science and Engineering, Vol. 16, 2000, pp. 169–200.

[20] Ferguson, D.—Siepmann, J.: Monte Carlo Methods in Chemical Physics. Kluwer Academic Publishers 1999, p. 467.

[21] Hernández, J. et al.: CMS Monte Carlo Production in the WLCG Computing Grid. Journal of Physics: Conference Series, 119:052019-052028, 2008.

[22] Herrea Sanz, J.: Programming Model for Grid Computing Infrastructures. Ph. D. thesis, Universidad Complutense de Madrid, Madrid 2009 (in Spanish).

[23] Herrera, J.—Huedo, E.—Montero, R.—Llorente, I. M.: Loosely-Coupled Loop Scheduling in Computational Grids. 20th International Parallel and Distributed Processing Symposium (IPDPS) 2006, p. 6.

[24] Hertz, D. B.: Risk Analysis in Capital Investment. Boston, MA: Harvard Business Review 1964.

[25] Hockney, R. W.—Jesshope, C. R.: Parallel Computers Two: Architecture, Programming and Algorithms. 1988.

[26] Huedo, E.—Montero, R.—Llorente, I. M.: The GridWay Framework for Adaptive Scheduling and Execution on Grids. Scalable Computing: Practice and Experience, Vol. 6, 2005, No. 8.

[27] Huedo, E.—Montero, R.—Llorente, I. M.: Evaluating the Reliability of Computational Grids from the end User's Point of View. Journal of Systems Architecture, Vol. 52, 2006, No. 12, pp. 727–736.

[28] Jang, S. H.–Wu, X.—Taylor, V.: Using Performance Prediction to Allocate Grid Resources. Technical report 2004.

[29] Jorgensen, W. L.—Tirado Rives, J.: Molecular Modeling of Organic and Biomolecular Systems Using BOSS and MCPRO. Journal of Computational Chemistry, Vol. 26, 2005, No. 16, pp. 1689–1700.

[30] Kitagawa, G.: Monte Carlo Filter and Smoother for non-Gaussian Nonlinear State Space Models. Journal of Computational and Graphical Statistics, Vol. 5, 1996, No. 1, pp. 1–25.

[31] Lepage, G. P.: Lattice QCD for Novices. ArXiv High Energy Physics – Lattice e-prints 2005.

[32] Li, C.—Li, L.: Utility-Based QoS Optimisation Strategy for Multi-Criteria Scheduling on the Grid. Journal of Parallel and Distributed Computing, Vol. 67, 2007, No. 2, pp. 142–153.

[33] Li, Y.—Mascagni, M.: Grid-Based Monte Carlo Application. Grid Computing – GRID 2002, pp. 13–24.

[34] Lister, G.: FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods, 1985.

[35] Lukkarinen, J.: Lattice Simulations of the Quantum Microcanonical Ensemble. ArXiv High Energy Physics – Theory e-prints 1997.

[36] Maigne, L.—Hill, D.—Calvat, P.—Breton, V.—Reuillon, R.—Legre, Y.—Donnarieix, Y.: Parallelization of Monte Carlo Simulations and Submission to a Grid Environment. Parallel Processing Letters, Vol. 14, 2004, No. 2, pp. 177–196.

[37] Malcolm, L.—Spaulding, L.—Anderson, E.—Tatsu, I.—Eoin, H.: Simulation of the Oil Trajectory and Fate in the Arabian Gulf from the Mina al Ahmadi Spil. Marine Environmental Research, Vol. 36, 1993, No. 2, pp. 79–115.

[38] Mascagni, M.—Li, Y.: Computational Infrastructure for Parallel, Distributed, and Grid-Based Monte Carlo Computations. Lecture Notes in Computer Sciences, Vol. 2907, 2003, pp. 39–52.

[39] Metropolis, N.—Rosenbluth, A.: Equation of State Calculations by Fast Computing Machines. The Journal of Chemical Physics, Vol. 6, 1953, No. 21, pp. 1087–1092.

[40] Metropolis, N.: The Beginning of the Monte Carlo Method. Los Alamos Science, Special Issue 1987.

[41] Montero, R.—Huedo, E.—Llorente, I. M.: Benchmarking of High Throughput Computing Applications on Grids. Parallel Computing, Vol. 32, 2006, No. 4, pp. 267–279.

[42] Pearson, K.: The Problem of the Random Walk. Nature, 72:294, 1905.

[43] Renshall, H.: New Time Units simplify procedures. CERN Computer Newsletter 2004.

[44] Rocha, L. F. O.—Tarragó Pinto, M. E.—Caliri, A.: The Water Factor in the Protein-Folding Problem. Brazilian Journal of Physics, Vol. 34, 2004, pp. 90–101.

[45] Rodríguez-Pascual, M.—Guasp, J.—Castejon Magana, P.—Rubio-Montero, A. J.—Llorente, I. M.—Mayo Garcia, R.: Improvements on the Fusion Code FAFNER2. IEEE Transactions on Plasma Science, Vol. 38, 2010, pp. 2102–2110.

[46] Roveda, R.—Goldstein, D. B.—Varghese, P. L.: Hybrid Euler/Direct Simulation Monte Carlo Calculation of Unsteady Slit Flow. Journal of Spacecraft and Rockets, Vol. 37, 2000, No. 6.

[47] Schopf, J. M.—Darcy, M.—Miller, N.—Pearlman, L.—Foster, I.—Kesselman, C.: Monitoring and Discovery in GT4: Functionality and Performance. Technical report 2005.

[48] Silva, D.–Cirne, W.—Brasileiro, F.: Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. Euro-Par 2003 Parallel Processing, pp. 169–180.

[49] Teubel, A.—Guasp, J.—Liniers, M.: Monte Carlo Simulations of NBI into the TJ-II Helical Axis Stellarator. Technical Report 4 1994.

[50] Tröger, P.—Domagalski, P.: Standardization of an API for Distributed Resource Management System. Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), pp. 619–626.

[51] Tzen, T. H.—Ni, L. M.: Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers. IEEE Transactions on Parallel and Distributed Systems, Vol. 4, 1993, pp. 87–98.

[52] Vázquez-Poletti, J. L.—Huedo, E.—Montero, R.—Llorente, I. M.: A Comparison Between two Grid Scheduling Philosophies: EGEE WMS and Grid Way. Multiagent and Grid Systems, Vol. 3, 2007, No. 4, pp. 429–439.

[53] Vazquez-Poletti, J. L.—Huedo, E.—Montero, R.—Llorente, I. M.: CD-HIT Workow Execution on Grids Using Replication Heuristics. Cluster Computing and the Grid 2008, 8th IEEE International Symposium on CCGRID '08, pp. 735–740, 2008.

[54] Swendsen, R. H.—Wang, J. S.: Replica Monte Carlo Simulation of Spin Glasses. Physical Review Letters, Vol. 57, 1986, No. 2, pp. 2607–2609.

[55] Wu, M.—Sun, X.: Grid Harvest Service: A Performance System of Grid Computing. Journal of Parallel and Distributed Computing, Vol. 66, 2006, No. 10, pp. 1322–1337.

[56] Yang, C.-T.—Chang, S.-C.: A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters. Proceedings of International Conference on Computer Science 2003, pp. 1079–1088.

[57] Yu, C.—Marinescu, D. C.: Algorithms for Divisible Load Scheduling of Data-Intensive Applications. Journal of Grid Computing, Vol. 8, 2010, No. 1, pp. 133–155.

[58] Zhang, Y.—Kihara, D.—Skolnick, J.: Local Energy Landscape Flattening: Parallel Hyperbolic Monte Carlo Sampling of Protein Folding. Proteins, Vol. 48, 2002, No. 2, pp. 192–201.

Manuel Rodríguez-Pascual has a Master Degree in computing sciences from the Universidad Complutense de Madrid. Presently he is a Ph. D. candidate at CIEMAT, where he is involved in several international R & D projects. His interests are in the areas of application optimization for grid computing, scheduling and middleware development.

Ignacio M. Llorente has a graduate degree in physics (B. Sc. in physics and M. Sc. in computer science), a Ph. D. in physics (Program in computer Science) and an Executive Master in business administration. He has about 15 years of research experience in the field of high-performance parallel and distributed computing, grid computing and virtualization. Currently, he is a Full Professor in computer architecture and technology at Universidad Complutense de Madrid, where he leads the Distributed Systems Architecture Group.

Rafael Mayo-García is Ph. D. in physics by the Universidad Complutense de Madrid-UCM, Madrid (Spain) since 2004 and has obtained several postdoctoral fellowships such as Marie Curie and Juan de la Cierva Grants. He has been working as Researcher at UCM in experimental and Computational Plasma Physics. Since 2005 he works at CIEMAT (Madrid) in the ICT Division on Supercomputing and Grid developments. He belongs to the Spanish Royal Society of Physics and the Latin American Bioinformatics Society. He is the author of more than 60 publications and conference presentations and has been involved in 20 international and national R & D projects where he has also held managerial and coordinating activities.