CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Controlling fairness and task granularity in distributed, online, non-clairvoyant workflow executions

Rafael Ferreira da Silva[1,2*], Tristan Glatard[1,3], Frédéric Desprez[4]

[1] *University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France*
[2] *University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA*
[3] *McConnell Brain Imaging Centre, Montreal Neurological Institute, McGill University, Canada*
[4] *INRIA, University of Lyon, LIP, ENS Lyon, Lyon, France*

SUMMARY

Distributed computing infrastructures are commonly used for scientific computing, and science gateways provide complete middleware stacks to allow their transparent exploitation by end-users. However, administrating such systems manually is time-consuming and sub-optimal due to the complexity of the execution conditions. Algorithms and frameworks aiming at automating system administration must deal with online and non-clairvoyant conditions, where most parameters are unknown and evolve over time. We consider the problem of controlling task granularity and fairness among scientific workflows executed in these conditions. We present two self-managing loops monitoring the fineness, coarseness, and fairness of workflow executions, comparing these metrics to thresholds extracted from knowledge acquired in previous executions, and planning appropriate actions to maintain these metrics to appropriate ranges. Experiments on the European Grid Infrastructure show that our task granularity control can speed-up executions up to a factor of 2, and that our fairness control reduces slowdown variability by 3 to 7 compared to first-come-first-served. We also study the interaction between granularity control and fairness control: our experiments demonstrate that controlling task granularity degrades fairness, but that our fairness control algorithm can compensate this degradation.

Received . . .

KEY WORDS: Distributed computing infrastructures, task granularity, fairness, scientific workflows, online conditions, non-clairvoyant conditions.

1. INTRODUCTION

Distributed computing infrastructures (DCI) such as grids and clouds are becoming daily instruments of scientific research, commonly exploited by science gateways [1] to offer transparent service to end-users. However, the large scale of these infrastructures, their heterogeneity and the complexity of their middleware stacks make them prone to software and hardware errors manifested by uncompleted or under-performing executions. Substantial system administration efforts need to be invested to improve quality of service, still at a non-optimal level.

Autonomic computing [2], in particular self-management, refers to the set of methods and algorithms addressing this challenge with control loops based on Monitoring, Analysis, Planning, Execution and Knowledge (MAPE-K). These algorithms have to cope with *online* conditions, where the platform load and infrastructure status constantly change, and *non-clairvoyant* conditions, where most of the applications' and infrastructure's characteristics are unknown. The interactions between

*Correspondence to: 4676 Admiralty Way, Suite 1001, 90292, Marina del Rey, CA, USA. E-mail: rafsilva@isi.edu

self-managing loops have to be properly studied to check that their individual objectives do not conflict.

This paper focuses on the control of task granularity and fairness among executions described as scientific workflows [3] in a science gateway. Task granularity is commonly optimized by grouping, or clustering, tasks of a workflow execution to minimize the impact of data transfers, task queuing times, and other overheads [4–13]. Group sizes have to be carefully tuned to avoid harmful parallelism losses. Fairness is achieved when resources are allocated to scientific workflow executions in proportion to their requirements rather than as a result of race conditions. A way to quantify the fairness obtained for a particular workflow execution is to determine the slowdown compared to an ideal situation where it would run alone on the whole infrastructure [14, 15]. Other fairness metrics can also be used, as for instance in [16].

We describe two self-managing loops to control task granularity and fairness of scientific workflows executed in online, non-clairvoyant conditions on DCIs. These algorithms periodically evaluate metrics estimating the coarseness, fineness, and fairness of workflow executions. When these metrics exceed thresholds determined from historical executions, pre-defined actions are implemented: (i) tasks are grouped or de-grouped to adjust granularity (ii) tasks are re-prioritized to improve fairness. These two loops were independently introduced and evaluated in [17] and [18]. In this work, we complement these previous presentations by studying the interaction between the two algorithms. Adjusting task granularity obviously impacts resource allocation, therefore fairness among executions. We approach this issue from an experimental angle, testing the following hypotheses:

- **H1**: the granularity control loop reduces fairness among executions;
- **H2**: the fairness control loop avoids this reduction.

The paper is organized as follows. Section 2 reviews the work related to task grouping and fairness control. Section 3 describes the two self-managing loops to control task granularity and fairness. Section 4 presents experiments evaluating our two control processes independently, and their interaction. Experiments are performed in real conditions, on the production system of the European Grid Infrastructure (EGI) accessed through the Virtual Imaging Platform (VIP [19]).

2. RELATED WORK

Task grouping. The low performance of *fine-grained* tasks is a common problem in widely distributed platforms where the scheduling overhead and queuing times are high, such as grid and cloud systems. Several works have addressed the control of task granularity of bag of tasks. For instance, Muthuvelu et al. [4] proposed an algorithm to group bag of tasks based on their granularity size – defined as the processing time of the task on the resource. Resources are ordered by their decreasing values of capacity (in MIPS) and tasks are grouped up to the resource capacity. This process continues until all tasks are grouped and assigned to resources. Then, Keat et al. [5] and Ang et al. [6] extended the work of Muthuvelu et al. by introducing bandwidth in the scheduling framework to enhance the performance of task scheduling. Resources are sorted in decreasing order of bandwidth, then assigned to grouped tasks downward ordered by processing requirement length. The size of a grouped task is determined from the task cost in millions instructions (MI). Later, Muthuvelu et al. [7] extended [4] to determine task granularity based on QoS requirements, task file size, estimated task CPU time, and resource constraints. Meanwhile, Liu & Liao [8] proposed an adaptive fine-grained job scheduling algorithm (AFJS) to group lightweight tasks according to processing capacity (in MIPS) and bandwidth (in Mb/s) of the current available resources. Tasks are sorted in decreasing order of MI, then clustered by a greedy algorithm. To accommodate with resource dynamicity, the grouping algorithm integrates monitoring information about the current availability and capability of resources. Afterwards, Soni et al. [9] proposed an algorithm to group lightweight tasks into coarse-grained tasks (GBJS) based on processing capability, bandwidth, and memory-size of the available resources. Tasks are sorted into ascending order of required computational power, then, selected in first come first serve order to be grouped according to the

DOI: 10.1002/cpe

capability of the resources. Zomaya and Chan [10] studied limitations and ideal control parameters of task clustering by using genetic algorithms. Their algorithm performs task selection based on the earliest task start time and task communication costs; it converges to an optimal solution of the number of clusters and tasks per cluster. Although the reviewed works significantly reduce communication and processing time, neither of them are non-clairvoyant and online at the same time. Recently, Muthuvelu et al. [11, 12] proposed an online scheduling algorithm to determine the task granularity of compute-intensive bag-of-tasks applications. The granularity optimization is based on task processing requirements, resource-network utilisation constraint (maximum time a scheduler waits for data transfers), and users QoS requirements (user's budget and application deadline). Submitted tasks are categorised according to their file sizes, estimated CPU times, and estimated output file sizes, and arranged in a tree structure. The scheduler selects a few tasks from these categories to perform resource benchmarking. Tasks are grouped according to seven objective functions of task granularity, and submitted to resources. The process restarts upon task arrival. In a collaborative work [13], we presented three balancing methods to address the load balancing problem when clustering scientific workflow tasks. We defined three imbalance metrics to quantitative measure workflow characteristics based on task runtime variation (HRV), task impact factor (HIFV), and task distance variance (HDV). Although these are online approaches, the solutions are still clairvoyant.

Fairness. Fairness among scientific workflow executions has been addressed in several studies considering the scheduling of multiple scientific workflows, but to the best of our knowledge, no algorithm was proposed in a non-clairvoyant and online case. For instance, Zhao and Sakellariou [14] address fairness based on the slowdown of Directed Acyclic Graph (DAG); they consider a clairvoyant problem where the execution time and the amount of data transfers are known. Similarly, N'Takpé and Suter [20] propose a mapping procedure to increase fairness among parallel tasks on multi-cluster platforms; they address an offline and clairvoyant problem where tasks are scheduled according to one of the following three characteristics: critical path length, maximal exploitable task parallelism, or amount of work to execute. Casanova et al. [15] evaluate several scheduling online algorithms of multiple parallel task graphs (PTGs) on a single, homogeneous cluster. Fairness is measured through the maximum stretch (a.k.a. slowdown) defined by the ratio between the PTG execution time on a dedicated cluster, and the PTG execution time in the presence of competition with other PTGs. Hsu et al. [21] propose an online HEFT-like algorithm to schedule multiple workflows; they address a clairvoyant problem where tasks are ranked based on the length of their critical path, and tasks are mapped to the resources with the earliest finish time. Sommerfeld and Richter [22] present a two-tier HEFT-based grid workflow scheduler with predictions of input-queue waiting times and task execution times; fairness among workflow tasks is addressed by preventing HEFT to assign the highest ranks to the first tasks regardless of their originating workflows. Hirales-Carbajal et al. [23] schedule multiple parallel workflows on a grid in a non-clairvoyant but offline context, assuming dedicated resources. Their multi-stage scheduling strategies consist of task labeling and adaptive allocation, local queue prioritization and site scheduling algorithm. Fairness among workflow tasks is achieved by task labeling based on task run time estimation. Recently, Arabnejad and Barbosa [24] proposed an algorithm addressing an online but clairvoyant problem where tasks are assigned to resources based on their rank values; task rank is determined from the smallest remaining time among all remaining tasks of the workflow, and from the percentage of remaining tasks. Finally, in their evaluation of non-preemptive task scheduling, Sabin et al. [25] assess fairness by assigning a fair start time to each task, defined by the start time of the task on a complete simulation of all tasks whose queue time is lower than that one. Any task which starts after its fair start time is considered to have been treated unfairly. Results are trace-based simulations over a period of one month, but the study is performed in a clairvoyant context. Skowron and Rzadca [26] proposed an online and non-clairvoyant algorithm to schedule sequential jobs on distributed systems. They consider a non-clairvoyant model where job's processing time is unknown until the job completes. However, they assume that resources are homogeneous (what is not the case on grid computing). In contrast, our method considers resource

DOI: 10.1002/cpe

performance, the execution of concurrent activities, and task dependency in scientific workflow executions.

3. METHODS

Scientific workflows consist of activities linked by data and control dependencies. Activities are bound to an application executable. At runtime, they are iterated on the data to generate tasks. A workflow activity is said active if it has at least one waiting (queued) or running task. Tasks cannot be pre-empted, i.e. only the tasks in status waiting can be modified. More details about our workflow model are available in [27]. Tasks related to an activity are assumed independent from each other, yet with similar cost (bag of tasks). It means that if these tasks were executed in the exact same conditions, they would be of identical duration.

The following sub-sections describe our task granularity and fairness control processes. In all the algorithms, η describes the degree quantifying the controlled variable (fineness, coarseness, or unfairness), and τ describes the threshold beyond which actions have to be triggered. Indexes refer to the controlled variable: f for fineness, c for coarseness, and u for unfairness. For instance, η_f is the fineness degree, and τ_u is the unfairness threshold.

3.1. Task Granularity Control Process

Algorithm 1 describes our task granularity control composed of two processes: (i) the fineness control process groups too fine task groups for which the fineness degree η_f is greater than threshold τ_f, and (ii) the coarseness control process de-groups too coarse task groups for which the coarseness degree η_c is greater than threshold τ_c. This subsection describes how η_f, η_c, τ_f and τ_c are determined, and details the grouping and de-grouping algorithms.

Algorithm 1 Main loop for granularity control

```
1:  input: n waiting tasks
2:  create n 1-task groups T_i
3:  while there is an active task group do
4:      wait for timeout or task status change
5:      determine fineness degree η_f
6:      if η_f > τ_f then
7:          group task groups using Algorithm 2
8:      end if
9:      determine coarseness degree η_c
10:     if η_c > τ_c then
11:         degroup coarsest task groups
12:     end if
13: end while
```

Measuring fineness: η_f. Let n be the number of waiting tasks in a scientific workflow activity, and m the number of task groups. Initially, 1 group is created for each task ($n = m$). T_i is the set of tasks in group i, and n_i is the number of tasks in T_i. Groups are a partition of the set of waiting tasks: $\bigcap_{i \neq j} T_j = \emptyset$ and $\sum_{i=1}^{m} n_i = n$. The activity fineness degree η_f is the maximum of all group fineness degrees f_i:

$$\eta_f = \max_{i \in [1,m]} (f_i). \tag{1}$$

High fineness degrees indicate fine granularities. We use a $\max$ operator in this equation to ensure that *any* task group with a too fine granularity will be detected. The fineness degree f_i of group i is defined as:

$$f_i = d_i \cdot r_i, \tag{2}$$

DOI: 10.1002/cpe

where r_i is the queuing ratio (described below), and d_i is the ratio between the transfer time of the input data shared among all tasks in the activity, and the total execution time of the group:

$$d_i = \frac{\tilde{t}_{\_shared}}{\tilde{t}_{\_shared} + n_i(\tilde{t} - \tilde{t}_{\_shared})},$$

where $\tilde{t}_{\_shared}$ is the median transfer time of the input data shared among all tasks in the activity, and $\tilde{t}$ is the sum of its median task phase durations corresponding to application setup, input data transfer, application execution and output data transfer: $\tilde{t} = \tilde{t}_{\_setup} + \tilde{t}_{\_input} + \tilde{t}_{\_exec} + \tilde{t}_{\_output}$. Median values $\tilde{t}_{\_shared}$ and $\tilde{t}$ are computed from values measured on completed tasks. When less than 2 tasks are completed, medians remain undefined and the control process is inactive. This online estimation makes our process non-clairvoyant with respect to the task duration which is progressively estimated as the workflow activity runs. aware that using the median may be inaccurate. However, without a model of the applications' execution time, we must rely on observed task durations. Using the whole time distribution (or at least its few first moments) may be more accurate but it would complexify the method.

In equation 2, r_i is the ratio between the max of the task current queuing times q_i in the group (measured for each task individually), and the total round-trip time (queuing+execution) of the group:

$$r_i = \frac{\max_{j\in[1,n_i]} q_j}{\max_{j\in[1,n_i]} q_j + \tilde{t}_{\_shared} + n_i(\tilde{t} - \tilde{t}_{\_shared})}$$

The group queuing time is the max of all task queuing times in the group, while the group execution time is the time to transfer shared input data plus the time to execute all task phases in the group except for the transfers of shared input data. Note that d_i, r_i, and therefore f_i and η_f are in $[0, 1]$. η_f tends to 0 when there is little shared input data among the activity tasks or when the task queuing times are low compared to the execution times; in both cases, grouping tasks is indeed useless. Conversely, η_f tends to 1 when the transfer time of shared input data becomes high, and the queuing time is high compared to the execution time; grouping is needed in this case.

Thresholding fineness: τ_f. The threshold value for η_f separates configurations where the activity's fineness is acceptable ($\eta_f \leq \tau_f$) from configurations where the activity is too fine ($\eta_f > \tau_f$). We determine τ_f from execution traces, inspecting the modes of the distribution of η_f. Values of η_f in the highest mode of the distribution, i.e. which are clearly separated from the others, will be considered too fine. Threshold value determined from traces are likely to be dependent on the execution conditions covered by the traces. In particular, traces have to be extracted from the platform where the methods will be implemented.

The traces were collected from VIP [19] between January 2011 and April 2012, made available through the science-gateway workload archive [28]. The data set contains $680,988$ tasks (including resubmissions and replications) linked to activities of $2,941$ scientific workflows executed by 112 users; task average waiting time is about 36 min. Applications deployed in VIP are described as scientific workflows executed using the MOTEUR workflow engine [29]. Resource provisioning and task scheduling are provided by DIRAC [30] using so-called "pilot jobs". Resources are provisioned online with no advance reservations. Tasks are executed on the biomed virtual organization (VO) of the European Grid Infrastructure (EGI)[†] which has access to some 150 computing sites worldwide and to 120 storage sites providing approximately 4 PB of storage. Figure 1 (left) shows the distribution of sites per country supporting the biomed VO.

The fineness degree η_f was computed after each event found in the data set. Figure 1 (right) shows the histogram of these values. The histogram appears bimodal, which indicates that η_f separates the platform configurations in two distinct groups, separated by the threshold value $\tau_f = 0.55$. In our case, this threshold value is visually determined from the histogram. Simple clustering methods

[†]http://www.egi.eu

DOI: 10.1002/cpe

such as k-means can be used to determine the threshold value automatically [31]. We consider that these groups correspond to "acceptable fineness" (lowest mode) and "too fine granularity" (highest mode). For $\eta_f \geq 0.55$, task grouping will be triggered.

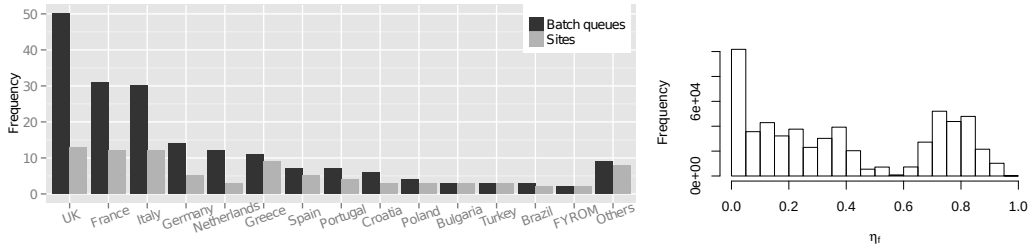


Figure 1. Distribution of sites and batch queues per country in the biomed VO (January 2013) (*left*) and histogram of fineness incident degree sampled in bins of 0.05 (*right*).

Task grouping. Algorithm 2 describes our task grouping mechanism. Groups with $f_i > \tau_f$ are grouped pairwise until $\eta_f \leq \tau_f$ or until the amount of waiting groups Q is smaller or equal to the amount of running groups R. Although η_f ignores scattering (Eq. 1 uses a $\max$), the algorithm considers it by grouping tasks in all groups where $f_i > \tau_f$. Ordering groups by decreasing f_i values tends to equally distribute tasks among groups. The grouping process stops when $Q \leq R$ to avoid parallelism loss. This condition also avoids conflicts with the de-grouping process described above.

Algorithm 2 Task grouping

```
1: input: f_1 to f_m //group fineness degrees, sorted in decreasing order
2: input: Q, R // number of queued and running task groups
3: for i = 1 to m − 1 do
4:     j = i + 1
5:     while f_i > τ_f and Q > R and j ≤ m do
6:         if f_j > τ_f then
7:             Group all tasks of T_j into T_i
8:             Recalculate f_i using Equation 2
9:             Q = Q − 1
10:        end if
11:        j = j + 1
12:    end while
13:    i = j
14: end for
15: Delete all empty task groups
```

Measuring coarseness: η_c. Condition $Q > R$ used in Algorithm 2 ensures that all resources will be exploited *if the number of available resources is stationary.* In case the number of available resources decreases, the fineness control process may further reduce the number of groups. However, if the number of available resources increases, task groups may need to be de-grouped to maximize resource exploitation. This de-grouping is implemented by our coarseness control process.

The coarseness control process monitors the value of η_c defined as:

$$\eta_c = \frac{R}{Q+R}. \tag{3}$$

The threshold value τ_c is set to 0.5 so that $\eta_c > \tau_c \Leftrightarrow Q < R$.

When an activity is considered too coarse, its groups are ordered by increasing values of η_f and the first groups (i.e. the coarsest ones) are split until $\eta_c < \tau_c$. Note that de-grouping increases the number of queued tasks, therefore tends to reduce η_c.

DOI: 10.1002/cpe

3.2. Fairness Control Process

Algorithm 3 describes our fairness control process. Fairness is controlled by allocating resources to workflows according to their fraction of pending work. It is done by re-prioritising tasks in workflows where the unfairness degree η_u is greater than a threshold τ_u. This section describes how η_u and τ_u are computed, and details the re-prioritization algorithm.

Algorithm 3 Main loop for fairness control

```
1: input: m workflow executions
2: while there is an active workflow do
3:     wait for timeout or task status change in any workflow
4:     determine unfairness degree η_u
5:     if η_u > τ_u then
6:         re-prioritize tasks using Algorithm 4
7:     end if
8: end while
```

Measuring unfairness: η_u. Let m be the number of workflows with an active activity. The unfairness degree η_u is the maximum difference between the fractions of pending work:

$$\eta_u = W_{\max} - W_{\min}, \tag{4}$$

with $W_{\min} = \min\{W_i, i \in [1, m]\}$ and $W_{\max} = \max\{W_i, i \in [1, m]\}$. All W_i are in $[0, 1]$. For $\eta_u = 0$, we consider that resources are fairly distributed among all workflows; otherwise, some workflows consume more resources than they should. The fraction of pending work W_i of a workflow $i \in [1, m]$ is defined from the fraction of pending work $w_{i,j}$ of its n_i active activities:

$$W_i = \max_{j \in [1, n_i]} (w_{i,j}) \tag{5}$$

All $w_{i,j}$ are between 0 and 1. A high $w_{i,j}$ value indicates that the activity has a lot of pending work compared to the others. We define $w_{i,j}$ as:

$$w_{i,j} = \frac{Q_{i,j}}{Q_{i,j} + R_{i,j} P_{i,j}} \cdot \hat{T}_{i,j}, \tag{6}$$

where $Q_{i,j}$ is the number of waiting tasks in the activity, $R_{i,j}$ is the number of running tasks in the activity, $P_{i,j}$ is the performance of the activity, and $\hat{T}_{i,j}$ is its relative observed duration. $\hat{T}_{i,j}$ is defined as the ratio between the median duration $\tilde{t}_{i,j}$ of the completed tasks in activity j and the maximum median task duration among all active activities of all running workflows:

$$\hat{T}_{i,j} = \frac{\tilde{t}_{i,j}}{\max_{v \in [1,m], w \in [1, n_i^*]} (\tilde{t}_{v,w})} \tag{7}$$

where $\tilde{t}_{i,j}$ is computed as $\tilde{t}_{i,j} = \tilde{t}_{i,j}^{setup} + \tilde{t}_{i,j}^{input} + \tilde{t}_{i,j}^{exec} + \tilde{t}_{i,j}^{output}$. Medians are progressively estimated as tasks complete. At the beginning of the execution, $\hat{T}_{i,j}$ is initialized to 1 and all medians are undefined; when two tasks of activity j complete, $\tilde{t}_{i,j}$ is updated and $\hat{T}_{i,j}$ is computed with equation 7. In this equation, the $\max$ operator is computed only on $n_i^* \leq n_i$ activities with at least 2 completed tasks, i.e. for which $\tilde{t}_{i,j}$ can be determined.

In Eq. 6, the performance $P_{i,j}$ of an activity varies between 0 and 1. A low $P_{i,j}$ indicates that resources allocated to the activity have bad performance for the activity; in this case, the contribution of running tasks is reduced and $w_{i,j}$ increases. Conversely, a high $P_{i,j}$ increases the contribution of running tasks, therefore decreases $w_{i,j}$. For an activity j with k_j active tasks, we define $P_{i,j}$ as:

$$P_{i,j} = 2 \left(1 - \max_{u \in [1, k_j]} \left\{ \frac{t_u}{\tilde{t}_{i,j} + t_u} \right\} \right), \tag{8}$$

DOI: 10.1002/cpe

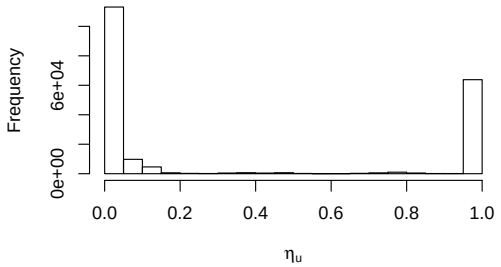


Figure 2. Histogram of the unfairness degree η_u sampled in bins of 0.05.

where $t_u = t_u^{setup} + t_u^{input} + t_u^{exec} + t_u^{output}$ is the sum of the estimated durations of task u's phases. Estimated task phase durations are computed as the max between the current elapsed time in the task phase (0 if the task phase has not started) and the median duration of the task phase. $P_{i,j}$ is initialized to 1, and updated using Eq. 8 only when at least 2 tasks of activity j are completed.

If all tasks perform as the median, i.e. $t_u = \tilde{t}_{i,j}$, then $\max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j}+t_u}\right\} = 0.5$ and $P_{i,j} = 1$. Conversely, if a task in the activity lasts significantly longer than the median, i.e. $t_u \gg \tilde{t}_{i,j}$, then $\max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j}+t_u}\right\} \approx 1$ and $P_{i,j} \approx 0$. This definition of $P_{i,j}$ considers that bad performance leads to a few tasks blocking the activity. Indeed, we assume that the scheduler does not deliberately favor any activity and that performance discrepancies are manifested by a few "unlucky" tasks slowed down by bad resources. Performance, in this case, has a relative definition: depending on the activity profile, it can correspond to CPU, RAM, network bandwidth, latency, or a combination of those. We admit that this definition of $P_{i,j}$ is a bit rough. However, under our non-clairvoyance assumption, estimating resource performance for the activity more accurately is hardly possible because (i) we have no model of the application, therefore task durations cannot be predicted from CPU, RAM or network characteristics, and (ii) network characteristics and even available RAM are shared among concurrent tasks running on the infrastructure, which makes them hardly measurable.

Thresholding unfairness: τ_u. Task prioritisation is triggered when the unfairness degree is considered critical, i.e $\eta_u > \tau_u$. Similarly to the fineness degree, the unfairness degree η_u was computed after each event found in the data set. Figure 2 shows the histogram of these values, where only $\eta_u \neq 0$ values are represented. This histogram is clearly bi-modal, which is a good property since it reduces the influence of τ_u. From this histogram, we choose $\tau_u = 0.2$, a threshold value that clearly separates the two modes. For $\eta_u > 0.2$, the execution is considered unfair and task prioritization is triggered. In this particular case, the two modes are so well separated that the choice of the threshold value is not critical: any value between 0.2 and 0.9 can safely be chosen as it will have only minor effect on mode classification.

Task prioritization. The action taken to cope with unfairness is to increase the priority of $\Delta_{i,j}$ waiting tasks for all activities j of workflow i where $w_{i,j} - W_{\min} > \tau_u$. Running tasks cannot be pre-empted. Task priority is an integer initialized to 1. $\Delta_{i,j}$ is determined so that $\tilde{w}_{i,j} = W_{min} + \tau_u$, where $\tilde{w}_{i,j}$ is the estimated value of $w_{i,j}$ after $\Delta_{i,j}$ tasks are prioritized. We approximate $\tilde{w}_{i,j}$ as:

$$\tilde{w}_{i,j} = \frac{Q_{i,j} - \Delta_{i,j}}{Q_{i,j} + R_{i,j}P_{i,j}}\hat{T}_{i,j},$$

which assumes that $\Delta_{i,j}$ tasks will move from status queued to running, and that the performance of new resources will be maximal. It gives:

DOI: 10.1002/cpe

Algorithm 4 Task re-prioritization

```
1: input: W_1 to W_m //fractions of pending works
2: maxPriority = max task priority in all workflows
3: for i=1 to m do
4:    if W_i − W_min > τ_u then
5:       for j=1 to a_i do
6:          //a_i is the number of active activities in workflow i
7:          if w_{i,j} − W_min > τ_u then
8:             Compute Δ_{i,j} from equation 9
9:             for p=1 to Δ_{i,j} do
10:               if ∃ waiting task q in activity j with priority ≤ maxPriority then
11:                  q.priority = maxPriority + 1
12:               end if
13:            end for
14:         end if
15:      end for
16:   end if
17: end for
```

$$\Delta_{i,j} = Q_{i,j} - \left\lfloor \frac{(\tau_u + W_{\min})(Q_{i,j} + R_{i,j}P_{i,j})}{\hat{T}_{i,j}} \right\rfloor, \tag{9}$$

where $\lfloor\rfloor$ rounds a decimal down to the nearest integer value.

Algorithm 4 describes our task re-prioritization. *maxPriority* is the maximal priority value in all workflows. The priority of $\Delta_{i,j}$ waiting tasks is set to *maxPriority+1* in all activities j of workflows i where $w_{i,j} - W_{min} > \tau_u$. Note that this algorithm takes into account scatter among W_i although η_u ignores it (see Eq. 4). Indeed, tasks are re-prioritized in *any* workflow i for which $W_i - W_{\min} > \tau_u$.

The method also accommodates online conditions. If a new workflow i is submitted, then $R_{i,j} = 0$ for all its activities and $\hat{T}_{i,j}$ is initialized to 1. This leads to $W_{max} = W_i = 1$, which increases η_u. If η_u goes beyond τ_u, then $\Delta_{i,j}$ tasks of activity j of workflow i have their priorities increased to restore fairness. Similarly, if new resources arrive, then $R_{i,j}$ increase and η_u is updated accordingly.

Table I shows a simple example of the interaction of the task granularity and fairness control loops. The example illustrates the scenario where the granularity control loop reduces fairness among executions, and the fairness control loops counterbalances this reduction.

4. EXPERIMENTS

The experiments presented hereafter evaluate the task granularity and fairness control processes and their interaction. In the first scenario, we evaluate fairness, fineness and coarseness control independently, under stationary and non-stationary loads. In the second scenario, we activate both control processes to test the two hypotheses already mentioned in the introduction:

- **H1**: the granularity control loop reduces fairness among executions;
- **H2**: the fairness control loop avoids this reduction.

4.1. Experiment conditions and metrics

The target computing platform for these experiments is the biomed VO of EGI where the traces used to determine τ_f and τ_u were acquired (see Section 3.1). To ensure resource limitation without flooding the production system, experiments are performed only on 3 sites of different countries. Tasks generated by MOTEUR are submitted using the DIRAC scheduler. We use MOTEUR 0.9.21, configured to resubmit failed tasks up to 5 times, and with the task replication mechanism described

DOI: 10.1002/cpe

Consider two identical workflows, each of which composed of one activity with 10 tasks initially split in 10 groups, and assume that task input data are shared among all tasks (i.e. $\tilde{t}_{\_shared} = \tilde{t}_{\_input}$). Let $\tilde{t} = 10$ and $\tilde{t}_{\_shared} = 7$ (in arbitrary time units) obtained from two completed task groups. At time t, we assume $R = 2$ and $Q = 6$ for both workflows with the following values for waiting task groups:

j	$\max_{k\in[1,n_j]} q_k$	d_j	r_j	f_j
5	50	0.70	0.83	0.58
6	48	0.70	0.82	0.58
7	45	0.70	0.81	0.57
8	43	0.70	0.81	0.57
9	41	0.70	0.80	0.56
10	40	0.70	0.80	0.56

Workflow 1 has the granularity control active, while Workflow 2 does not. For Workflow 1, Eq. 1 gives $\eta_f = 0.58$. As $\eta_f > \tau_f = 0.55$ and $Q > R$, the activity is considered too fine and task grouping is triggered.
Groups with $f_i > \tau_f$ are grouped pairwise until $\eta_f \leq \tau_f$ or $Q \leq R$:

j	$\max_{k\in[1,n_j]} q_k$	d_j	r_j	f_j
11 [5,6]	50	0.53	0.79	0.42
12 [7,8]	45	0.53	0.77	0.41
13 [9,10]	41	0.53	0.76	0.40

Groups 5 and 6, 7 and 8, and 9 and 10 are grouped into groups 11, 12, and 13 for Workflow 1, while Workflow 2 remains with 6 task groups.
Let's assume at time $t' > t$, the following values:

i	$Q_{i,1}$	$R_{i,1}$	$\tilde{t}_{i,1}$	$P_{i,1}$	$\hat{T}_{i,1}$	$W_i = w_{i,1}$
1	1	2	10	1.0	1.0	0.33
2	5	1	10	1.0	1.0	0.83

The configuration is clearly unfair since Workflow 2 has more tasks to compute.
Eq. 4 gives $\eta_u = 0.5$. As $\eta_u > \tau_u = 0.2$, the platform is considered unfair and task re-prioritization is triggered.
$\Delta_{2,1}$ tasks from Workflow 2 should be prioritized. According to Eq. 9:

$$\Delta_{2,1} = Q_{2,1} - \left\lfloor \frac{(\tau_u + W_1)(Q_{2,1} + R_{2,1}P_{2,1})}{\hat{T}_{2,1}} \right\rfloor = 5 - \left\lfloor \frac{(0.2+0.33)(5+1\cdot 1.0)}{1.0} \right\rfloor = 3$$

At time $t'' > t'$:

i	$Q_{i,1}$	$R_{i,1}$	$\tilde{t}_{i,1}$	$P_{i,1}$	$\hat{T}_{i,1}$	$W_i = w_{i,1}$
1	1	2	10	0.9	1.0	0.35
2	2	4	10	1.0	1.0	0.33

.

Now, $\eta_u = 0.02 < \tau_u$. The platform is considered fair and no action is performed.

Table I. Example of the interaction of the task granularity and fairness control loops.

in [32] activated. We use the DIRAC v6r6p2 instance provided by France-Grilles[‡], with a first-come, first-served policy imposed by submitting workflows with decreasing priority values for the fairness control process experiment. Four real medical simulation workflows are considered: `GATE` [33], `SimuBloch` [34], `FIELD-II` [35], and `PET-Sorteo` [36]; their main characteristics are summarized on Table II.

[‡]https://dirac.france-grilles.fr

Workflow		#Tasks	CPU time	Input	Output	t_{shared}
GATE	(CPU-intensive)	100	few minutes to one hour	∼115 MB	∼40 MB	-
SimuBloch	(data-intensive)	25	few seconds	∼15 MB	< 5 MB	90%
FIELD-II	(data-intensive)	122	few seconds to 15 minutes	∼208 MB	∼40 KB	∼40-60%
PET-Sorteo	(CPU-intensive)	1→80→1→80→1→1	∼10 minutes	∼20 MB	∼50 MB	∼50-80%

Table II. Workflow characteristics (→ indicate task dependencies).

DOI: 10.1002/cpe

For all experiments, control and tested executions are launched simultaneously to ensure similar grid conditions which may vary among repetitions because computing, storage, and network resources are shared with other users. To cover different grid conditions, experiments are repeated over a time period of 4 weeks.

Three different fairness metrics are used in the experiments. Firstly, the standard deviation of the makespan, written σ_m, is a straightforward metric which can be used when identical workflows are executed. Secondly, we define the unfairness μ as the area under the curve η_u during the execution:

$$\mu = \sum_{i=2}^{M} \eta_u(t_i) \cdot (t_i - t_{i-1}),$$

where M is the number of time samples until the makespan. This metric measures if the fairness process can indeed minimize η_u. Finally, the slowdown s of a completed workflow execution is defined by [20]:

$$s = \frac{M_{multi}}{M_{own}}$$

where M_{multi} is the makespan observed on the shared platform, and M_{own} is the estimated makespan if it was executed alone on the platform. In our conditions, M_{own} is estimated as:

$$M_{own} = \max_{p \in \Omega} \sum_{u \in p} t_u,$$

where Ω is the set of task paths in the workflow, and t_u is the measured duration of task u. This assumes that concurrent executions only impact task waiting time, not performance. For instance, network congestion or changes in performance distribution resulting from concurrent executions are ignored. Our third fairness metric is σ_s, the standard deviation of the slowdown.

4.2. Task granularity control

The granularity control process was implemented as a plugin of the MOTEUR workflow manager, receiving notifications about task status changes and task phase durations. The plugin then uses this data to group and de-group tasks according to Algorithm 1, where the timeout value is set to 2 minutes.

Two sets of experiments are conducted, under different load patterns. Experiment 1 evaluates the fineness control process under stationary load. It consists of separated executions of `SimuBloch`, `FIELD-II`, and `PET-Sorteo/emission`. A workflow activity using our task grouping mechanism (`Fineness`) is compared to a control activity (`No-Granularity`). Resource contention on the execution sites is maintained high and constant so that no de-grouping is required. Experiment 2 evaluates the interest of using the de-grouping control process under non-stationary load. It uses activity `FIELD-II`. An execution using both fineness and coarseness control (`Fineness-Coarseness`) is compared to an execution without coarseness control (`Fineness`) and to a control execution (`No-Granularity`). Executions are started under resource contention, but the contention is progressively reduced during the experiment. This is done by submitting a heavy workflow before the experiment starts, and canceling it when half of the experiment tasks are completed. As no online task modification is possible in DIRAC, we implemented task grouping by canceling queued tasks and submitting grouped tasks as a new task. For each grouped task resubmitted in the `Fineness` or `Fineness-Coarseness` executions, a task in the `No-Granularity` is randomly selected and resubmitted too to ensure equal race conditions for resource allocation (first-come, first-served policy), and that each execution faces the same re-submission overhead.

Experiment 1 (*stationary load*). Figure 3 shows the makespan of `SimuBloch`, `FIELD-II`, and `PET-Sorteo/emission` executions. `Fineness` yields a significant makespan reduction for all repetitions. Table III shows the makespan (M) values and the number of task groups. The task

DOI: 10.1002/cpe

grouping mechanism is not able to group all SimuBloch tasks in a single group because 2 tasks must be completed for the process to have enough information about the application (i.e. $\tilde{t}_{\_shared}$ and $\tilde{t}$ can be computed). This is a constraint of our non-clairvoyant conditions, where task durations cannot be determined in advance. FIELD-II tasks are initially not grouped, but as the queuing time becomes important, tasks are considered too fine and grouped. PET-Sorteo/emission is an intermediary case where only a few tasks are grouped. Results show that the task grouping mechanism speeds up SimuBloch and FIELD-II executions up to a factor of 2.6, and PET-Sorteo/emission executions up to a factor of 2.5.

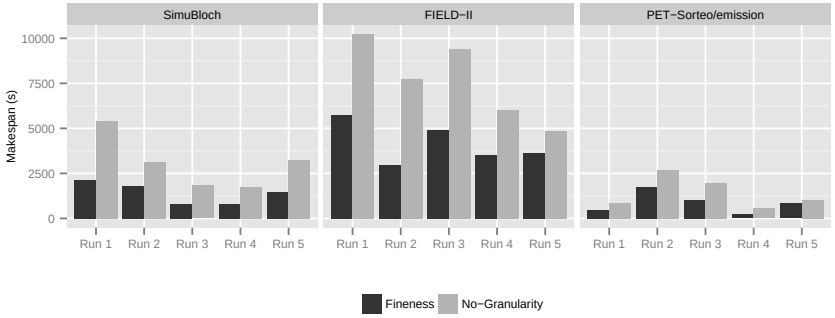


Figure 3. Experiment 1: makespan for Fineness and No-Granularity executions for the 3 workflow activities under stationary load.

		SimuBloch		FIELD-II		PET-Sorteo	
		M (s)	Groups	M (s)	Groups	M (s)	Groups
1	No-Granularity	5421	25	10230	122	873	80
	Fineness	2118	3	5749	80	451	57
2	No-Granularity	3138	25	7734	122	2695	80
	Fineness	1803	3	2982	75	1766	40
3	No-Granularity	1831	25	9407	122	1983	80
	Fineness	780	4	4894	73	1047	53
4	No-Granularity	1737	25	6026	122	552	80
	Fineness	797	6	3507	61	218	64
5	No-Granularity	3257	25	4865	122	1033	80
	Fineness	1468	4	3641	91	831	71

Table III. Experiment 1: makespan (M) and number of task groups for SimuBloch, FIELD-II and PET-Sorteo/emission executions for the 5 repetitions.

Experiment 2 (*non-stationary load*). Figure 4 shows the makespan (top) and evolution of task groups (bottom). Makespan values are reported in Table IV. In the first three repetitions, resources appear progressively during workflow executions. Fineness and Fineness-Coarseness speed up executions up to a factor of 1.5 and 2.1. Since Fineness does not benefit of newly arrived resources, it has a lower speed up compared to No-Granularity due to parallelism loss. In the two last repetitions, the de-grouping process in Fineness-Coarseness allows to reach similar performance than No-Granularity, while Fineness is penalized by its lack of adaptation: a slowdown of 20% is observed compared to No-Granularity. Table IV also shows the average queuing time values for Experiment 2. The linear correlation coefficient between the makespan and the average queuing time is 0.91, which indicates that the makespan evolution is indeed correlated to the evolution of the queuing time induced by the granularity control process.

Our task granularity control process works best under high resource contention, when the amount of available resources is stable or decreases over time (Experiment 1). Coarseness control can cope

DOI: 10.1002/cpe

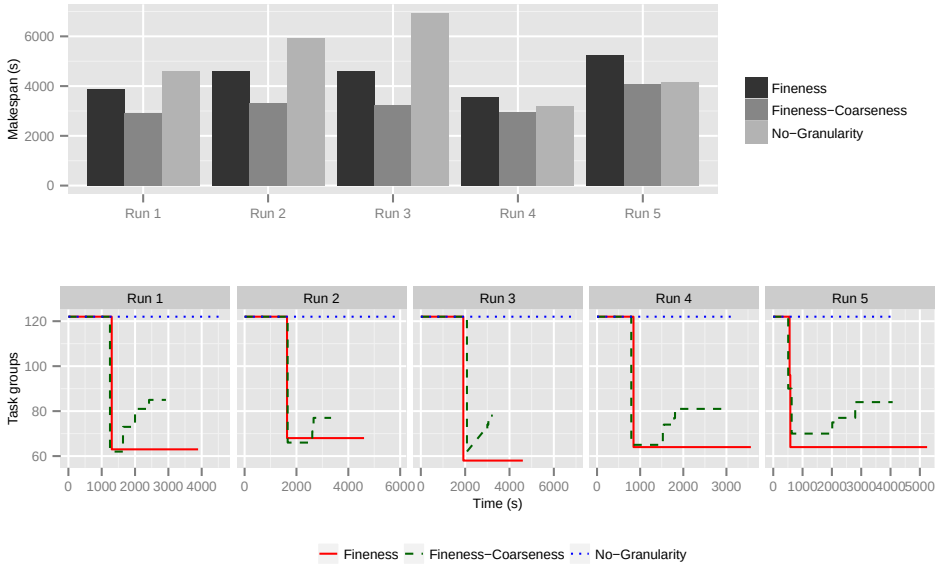


Figure 4. Experiment 2: makespan (top) and evolution of task groups (bottom) for `FIELD-II` executions under non-stationary load (resources arrive during the experiment).

	Run 1		Run 2		Run 3		Run 4		Run 5	
	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)
No-Granularity	4617	2111	5934	2765	6940	3855	3199	1863	4147	2295
Fineness	3892	2036	4607	2090	4602	2631	3567	1928	5247	2326
Fineness-Coarseness	2927	1708	3335	1829	3247	2091	2952	1586	4073	2197

Table IV. Experiment 2: makespan (M) and average queuing time ($\bar{q}$) for `FIELD-II` workflow execution for the 5 repetitions.

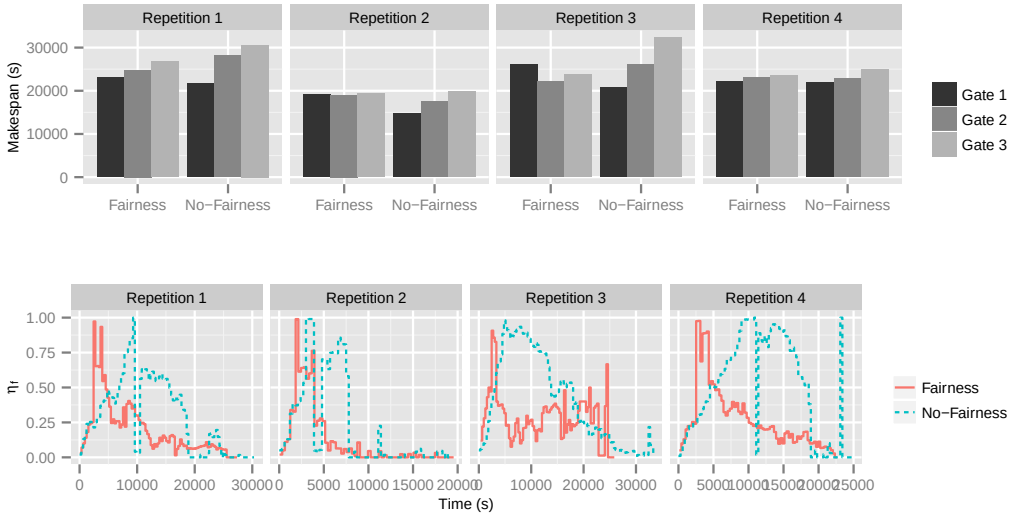
with soft increases in the number of available resources (Experiment 2), but fast variations remain difficult to handle. In the worst-case scenario, tasks are first grouped due to resource limitation, and resources suddenly appear once all task groups are already running. In this case the de-grouping algorithm has no group to handle, and granularity control penalizes the execution. Task pre-emption should be added to the method to address this scenario. In addition, our method is dependent on the capability to extract enough accurate information from completed tasks to handle active tasks using median estimates. This may not be the case for activities which execute only a few tasks.

4.3. Fairness control

Fairness control was implemented as a MOTEUR plugin receiving notifications about task and workflow status changes. Each workflow plugin forwards task status changes and $\tilde{t}_{i,j}$ values to a service centralizing information about all the active workflows. This service then re-prioritizes tasks according to Algorithms 3 and 4. As no online task modification is possible in DIRAC, we implemented task prioritization by canceling and resubmitting queued tasks with new priorities. This implementation decision adds an overhead to task executions. Therefore, the timeout value used in Algorithm 3 is set to 3 minutes.

Three experiments are conducted. Experiment 3 tests whether unfairness among *identical workflows* is properly addressed. It consists of three `GATE` workflows sequentially submitted, as users usually do in the platform. Experiment 4 tests if the performance of *very short workflow executions* is improved by the fairness mechanism. Its workflow set has three `GATE` workflows

DOI: 10.1002/cpe



Repetition 1				Repetition 2				Repetition 3				Repetition 4			
	$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$
NF	4666	1.03	8758	NF	2541	0.50	4154	NF	5791	2.10	13392	NF	1567	0.87	12283
F	1884	0.40	5292	F	167	0.07	2367	F	2007	0.84	7243	F	706	0.24	6070

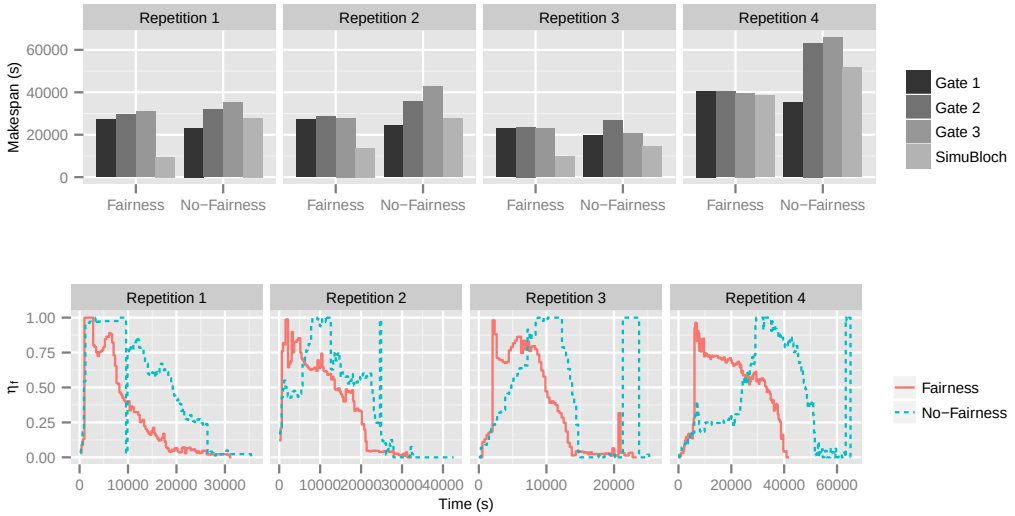
Figure 5. Experiment 3 (identical workflows). Top: comparison of the makespans; middle: unfairness degree η_u; bottom: standard deviation of the makespan (σ_m), standard deviation of the slowdown (σ_s), and unfairness (μ).

launched sequentially, followed by a `SimuBloch` workflow. Experiment 5 tests whether unfairness among *different workflows* is detected and properly handled. Its workflow set consists of a `GATE`, a `FIELD-II`, a `PET-Sorteo` and a `SimuBloch` workflow launched sequentially.

For each experiment, a workflow set using our fairness mechanism (`Fairness` – `F`) is compared to a control workflow set (`No-Fairness` – `NF`). `Fairness` and `No-Fairness` are launched simultaneously to ensure similar grid conditions. For each task priority increase in the `Fairness` workflow set, a task in the `No-Fairness` workflow set task queue is also prioritized to ensure equal race conditions for resource allocation. Experiment results are not influenced by the re-submission process overhead since both `Fairness` and `No-Fairness` experience the same overhead.

Experiment 3 (*identical workflows*). Figure 5 shows the makespan, unfairness degree η_u, makespan standard deviation σ_m, slowdown standard deviation σ_s and unfairness μ for the 4 repetitions. The difference among makespans and unfairness degree values are significantly reduced in all repetitions of `Fairness`. Both `Fairness` and `No-Fairness` behave similarly until η_u reaches the threshold value $\tau_u = 0.2$. Unfairness is then detected and the mechanism triggers task prioritization. Paradoxically, the first effect of task prioritization is a slight increase of η_u. Indeed, $P_{i,j}$ and $\hat{T}_{i,j}$, that are initialized to 1, start changing earlier in `Fairness` than in `No-Fairness` due to the availability of task duration values to compute $\tilde{t}_{i,j}$. Note that η_u reaches similar maximal values in both cases, but reaches them faster in `Fairness`. The fairness mechanism then manages to decrease η_u back under 0.2 much faster than it happens in `No-Fairness` when tasks progressively complete. Finally, slight increases of η_u are sporadically observed towards the end of the execution. This is due to task replications performed by MOTEUR: when new tasks are created, the fraction of pending work W increases, which has an effect on η_u. Quantitatively, the fairness mechanism reduces σ_m up to a factor of 15, σ_s up to a factor of 7, and μ by about 2.

DOI: 10.1002/cpe



Repetition 1			Repetition 2			Repetition 3			Repetition 4		
	σ_s	$\mu(s)$		σ_s	$\mu(s)$		σ_s	$\mu(s)$		σ_s	$\mu(s)$
NF	94.88	17269	NF	100.05	16048	NF	87.93	11331	NF	213.60	28190
F	15.95	9085	F	42.94	12543	F	57.62	7721	F	76.69	21355

Figure 6. Experiment 4 (very short execution). Top: comparison of the makespans; middle: unfairness degree η_u; bottom: unfairness (μ) and standard deviation of the slowdown (σ_s).

Experiment 4 (*very short execution*). Figure 6 shows the makespan, unfairness degree η_u, unfairness μ and slowdown standard deviation. In all cases, the makespan of the very short `SimuBloch` executions is significantly reduced for `Fairness`. The evolution of η_u is coherent with Experiment 1: a common initialization phase followed by an anticipated growth and decrease for `Fairness`. `Fairness` reduces σ_s up to a factor of 5.9 and unfairness up to a factor of 1.9.

Table V shows the execution makespan (m), average wait time ($\bar{w}$) and slowdown (s) values for the `SimuBloch` execution launched after the 3 `GATE`. As it is a non-clairvoyant scenario where no information about task execution time and future task submission is known, the fairness mechanism is not able to give higher priorities to `SimuBloch` tasks in advance. Despite that, the fairness mechanism speeds up `SimuBloch` executions up to a factor of 2.9, reduces task average wait time up to factor of 4.4 and reduces slowdown up to a factor of 5.9.

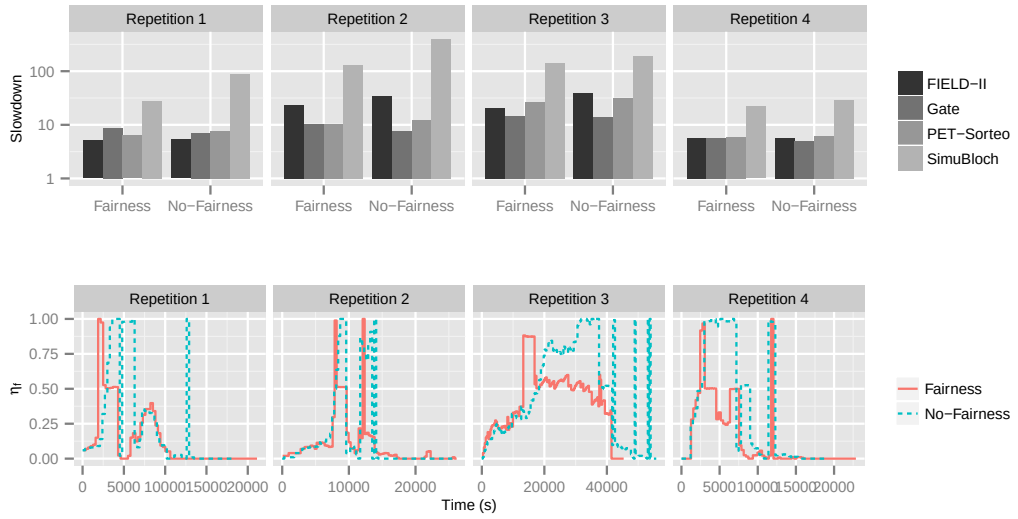
Run	Type	m (secs)	$\bar{w}$ (secs)	s
1	No-Fairness	27854	18983	196.15
	Fairness	9531	4313	38.43
2	No-Fairness	27784	19105	210.48
	Fairness	13761	10538	94.25
3	No-Fairness	14432	13579	182.68
	Fairness	9902	8145	122.25
4	No-Fairness	51664	47591	445.38
	Fairness	38630	27795	165.79

Table V. Experiment 4: `SimuBloch`'s makespan (m), average wait time ($\bar{w}$), and slowdown (s).

Experiment 5 (*different workflows*). Figure 7 shows slowdown, unfairness degree, unfairness μ and slowdown standard deviation σ_s for the 4 repetitions. `Fairness` slows down `GATE` while it speeds up all other workflows. This is because `GATE` is the longest and the first to be submitted; in `No-Fairness`, it is favored by resource allocation to the detriment of other workflows. The

DOI: 10.1002/cpe

evolution of η_u is similar to Experiments 3 and 4. σ_s is reduced up to a factor of 3.8 and unfairness up to a factor of 1.9.



Repetition 1			Repetition 2			Repetition 3			Repetition 4		
	σ_s	μ (s)		σ_s	μ (s)		σ_s	μ (s)		σ_s	μ (s)
NF	40.29	4443	NF	192.58	4004	NF	81.81	25173	NF	11.61	6613
F	10.52	2689	F	58.83	2653	F	60.56	18537	F	8.10	3303

Figure 7. Experiment 5 (different workflows). Top: comparison of the slowdown; middle: unfairness degree η_u; bottom: unfairness (μ) and standard deviation of the slowdown (σ_s).

In all 3 experiments, fairness optimization takes time to begin because the method needs to acquire information about the applications which are totally unknown when a workflow is launched. We could think of reducing the time of this information-collecting phase, e.g. by designing initialization strategies maximizing information discovery, but it couldn't be totally removed. Currently, the method works best for applications with a lot of short tasks because the first few tasks can be used for initialization, and optimization can be exploited for the remaining tasks. The worst-case scenario is a configuration where the number of available resources stays constant and equal to the number of tasks in the first submitted workflow: in this case, no action could be taken until the first workflow completes, and the method would not do better than first-come-first-served. Pre-emption of running tasks should be considered to address that.

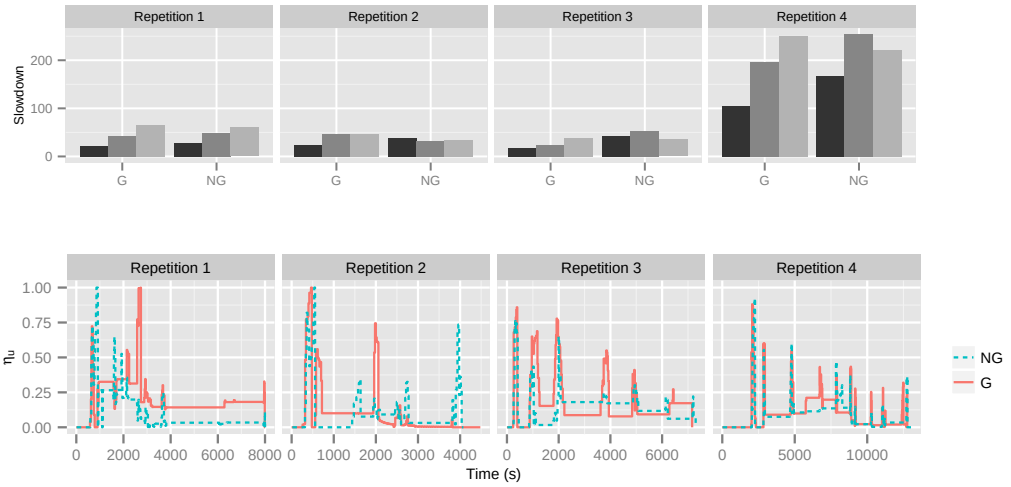
4.4. Interactions between task granularity and fairness control

In this section, we evaluate the interaction between both control processes in the same execution. For this experiment, we consider executions of a set of `SimuBloch` workflows under high resource contention. Two experiments are conducted. Experiment 6 tests whether the task granularity control process penalizes fairness among workflow executions (**H1**); Experiment 7 tests whether the fairness control process mitigates the unfairness created by the granularity control process (**H2**). For each experiment, a workflow set where one workflow uses the granularity control process (`Granularity - G`) is compared to a control workflow set (`No-Granularity - NG`). A workflow set consists of three `SimuBloch` workflows sequentially submitted. In the `Granularity` set, the first workflow has the granularity control process enabled, and the others do not. Experiment 6 has a fairness service which only measures the unfairness among workflow executions, but no action is triggered. In Experiment 7, task prioritization is triggered once unfairness is detected.

DOI: 10.1002/cpe

`Granularity` and `No-Granularity` are launched simultaneously to ensure similar grid conditions. Experiments 6 and 7 are also launched simultaneously. For each grouped task resubmitted in the `Granularity` execution, a task in the `No-Granularity` is resubmitted too in each experiment to ensure equal race conditions for resource allocation. Similarly, for each task prioritized in Experiment 7, a task in the Experiment 6 is also prioritized to ensure equal race conditions. Again, experiment results are not influenced by the submission process overhead since both `Granularity` and `No-Granularity` of both experiments experience the same overhead. Therefore, performance results obtained in Experiment 6 can be compared to the ones obtained in Experiment 7, and vice-versa.

Experiment 6 (*granularity without fairness*). Figure 8 shows the slowdown, unfairness degree η_u, makespan standard deviation σ_m, slowdown standard deviation σ_s, and unfairness μ for the 4 repetitions. Both `Granularity` and `No-Granularity` executions behave similarly until η_f reaches the threshold value $\tau_f = 0.55$. Tasks are considered too fine and the mechanism triggers task grouping. In all cases, the slowdown of the workflow executed with granularity (the first one on the `Granularity` set) is the lowest, i.e. its execution benefits of the grouping mechanism by saving waiting times and data transfers of shared input data. In the workflow set where the granularity control process is enabled, the unfairness value is up to a factor of 2 higher when compared to the workflow set where the granularity is disabled (`No-Granularity`). Hypothesis **H1** is therefore confirmed.

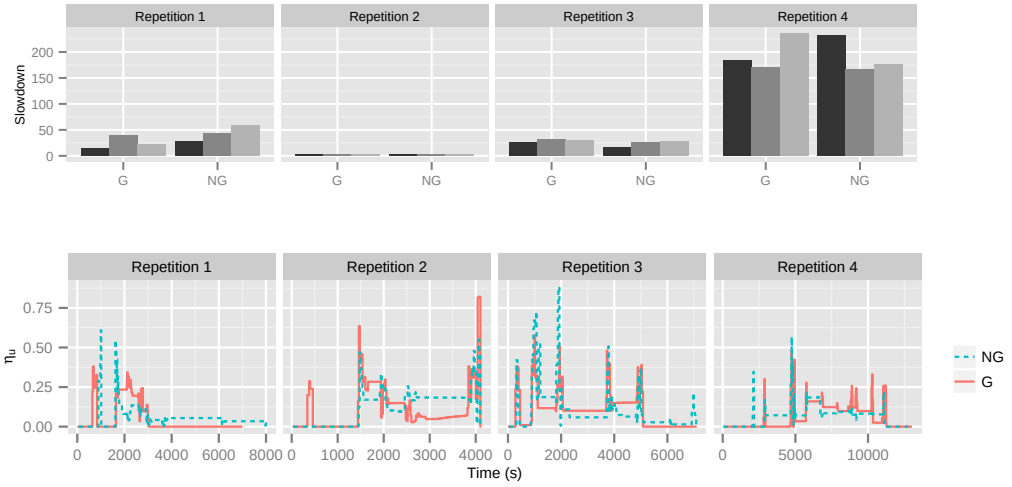


Repetition 1				Repetition 2				Repetition 3				Repetition 4			
	σ_m(s)	σ_s	μ(s)		σ_m(s)	σ_s	μ (s)		σ_m(s)	σ_s	μ (s)		σ_m(s)	σ_s	μ (s)
NG	2446	16.55	823	NG	15	3.24	503	NG	1273	8.91	998	NG	1364	38.70	1040
G	2962	21.94	1638	G	1015	13.15	527	G	2566	11.01	1212	G	2017	73.09	1250

Figure 8. Experiment 6 (granularity without fairness). Top: comparison of the slowdowns; middle: unfairness degree η_u; bottom: unfairness and standard deviation of the makespan and slowdown.

Experiment 7 (*granularity with fairness*). Figure 9 shows the slowdown, unfairness degree η_u, makespan standard deviation σ_m, slowdown standard deviation σ_s, and unfairness μ for the 4 repetitions. Both `Granularity` and `No-Granularity` executions have similar unfairness values which confirms **H2**. The same behavior is observed in σ_m and σ_s for repetitions 1, 3, and 4. This is not the case of repetition 2, in which resources suddenly appeared while tasks were being grouped. This resulted in parallelism loss for some workflow executions while the others were less impacted.

DOI: 10.1002/cpe



Repetition 1				Repetition 2				Repetition 3				Repetition 4			
	σ_m(s)	σ_s	μ(s)		σ_m(s)	σ_s	μ (s)		σ_m(s)	σ_s	μ (s)		σ_m(s)	σ_s	μ (s)
NG	2848	15.41	445	NG	78	0.11	501	NG	2758	6.11	718	NG	1044	35.37	826
G	2384	13.47	335	G	456	0.23	455	G	1674	3.11	712	G	1028	34.53	754

Figure 9. Experiment 7 (granularity with fairness). Top: comparison of the slowdowns; middle: unfairness degree η_u; bottom: unfairness and standard deviation of the makespan and slowdown.

5. CONCLUSION

We described two self-managing algorithms to control task granularity and fairness of scientific workflow executions. The fineness of workflow activities is defined from queue waiting time and estimated data transfer time of shared input data, their coarseness is measured based on the ratio of the number of queued tasks related to the number of running tasks, and fairness among scientific workflow executions is quantified from the fraction of pending work in workflow executions. All these metrics are computed online, estimating task durations in a workflow activity with the median duration of previous executions. Experiments conducted on the European Grid Infrastructure demonstrate the relevance of these algorithms to speed-up workflow execution by saving data transfers and queuing time, and to improve fairness among workflow executions in a science gateway. Experiments testing the interaction between granularity and fairness control showed that controlling granularity degrades fairness, but that our fairness control algorithm compensates this degradation. The methods were implemented in the Virtual Imaging Platform, and we plan to use them in production. We believe that they are generic enough to be re-used in other similar platforms.

Future work could target the initialization phase of the self-managing loops. Currently, when a scientific workflow starts, no information about the duration of its tasks on the infrastructure is available, which makes the methods completely blind until the first tasks complete. One approach could be to initialize task durations based on historical information. A sensitivity analysis on the influence of each terms of our metrics (d_i and r_i for granularity; $T_{i,j}$ and $P_{i,j}$ for the fairness) could also be done along with a comparison with scheduling algorithms such as Fair-Share [37]. Finally, the interaction between control loops could also be explored analytically, identifying conditions on the metrics (η_u, η_f, and η_c) to avoid conflicting objectives.

6. ACKNOWLEDGMENT

We thank the European Grid Infrastructure and its supporting National Grid Initiatives, in particular France-Grilles, for providing the technical support, computing and storage facilities. Results obtained in this paper were computed on the biomed virtual organization of the European Grid

DOI: 10.1002/cpe

Infrastructure (http://www.egi.eu). This work was funded by the French National Agency for Research under grant ANR-09-COSI-03 “VIP”. The research leading to this publication has also received funding from the EC FP7 Programme under grant agreement 312579 ER-flow = Building an European Research Community through Interoperable Workflows and Data. This work was performed within the framework of the LABEX PRIMES (ANR-11-LABX-0063) of Université de Lyon, within the program ”Investissementsd’Avenir” (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

REFERENCES

1. Kiss T. Science Gateways for the Broader Take-up of Distributed Computing Infrastructures. *Journal of Grid Computing* Nov 2012; **10**(4):599–600.
2. Kephart JO, Chess DM. The vision of autonomic computing. *Computer* Jan 2003; **36**(1):41–50.
3. Taylor I, Deelman E, Gannon D. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 2006.
4. Muthuvelu N, Liu J, Soe NL, Venugopal S, Sulistio A, Buyya R. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. *Proceedings of the 2005 Australasian workshop on Grid computing and e-research - Volume 44*, ACSW Frontiers ’05, Australian Computer Society, Inc.: Darlinghurst, Australia, Australia, 2005; 41–48.
5. Ng WK, Ang TF, Ling TC, Liew CS. Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing. *Malaysian Journal of Computer Science* 2006; **19**.
6. Ang T, Ng W, Ling T, Por L, Liew C. A bandwidth-aware job grouping-based scheduling on grid environment. *Information Technology Journal* 2009; **8**:372–377.
7. Muthuvelu N, Chai I, Eswaran C. An adaptive and parameterized job grouping algorithm for scheduling grid jobs. *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*, vol. 2, 2008; 975 –980.
8. Liu Q, Liao Y. Grouping-based fine-grained job scheduling in grid computing. *ETCS ’09*, vol. 1, 2009; 556 –559.
9. Soni VK, Sharma R, Mishra MK. Grouping-based job scheduling model in grid computing. *World Academy of Science, Engineering and Technology* 2010; **41**:781–784.
10. Zomaya A, Chan G. Efficient clustering for parallel tasks execution in distributed systems. *18th IPDPS*, 2004; 167–174.
11. Muthuvelu N, Chai I, Chikkannan E, Buyya R. On-line task granularity adaptation for dynamic grid applications. *Algorithms and Architectures for Parallel Processing*, *LNCS*, vol. 6081. Springer, 2010; 266–277.
12. Muthuvelu N, Vecchiola C, Chai I, Chikkannan E, Buyya R. Task granularity policies for deploying bag-of-task applications on global grids. *Future Generation Computer Systems* 2013; **29**(1):170 – 181, doi:10.1016/j.future.2012.03.022. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.
13. Chen W, Ferreira da Silva R, Deelman E, Sakellariou R. Balanced task clustering in scientific workflows. *eScience (eScience), 2013 IEEE 9th International Conference on*, 2013; 188–195, doi:10.1109/eScience.2013.40.
14. Zhao H, Sakellariou R. Scheduling multiple DAGs onto heterogeneous systems. *20th International Parallel and Distributed Processing Symposium*, IPDPS 2006, 2006, doi:10.1109/IPDPS.2006.1639387.
15. Casanova H, Desprez F, Suter F. On cluster resource allocation for multiple parallel task graphs. *J. of Par. and Dist. Computing* 2010; **70**(12):1193 – 1203.
16. Kleban S, Clearwater S. Fair share on high performance computing systems: what does fair really mean? *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, 2003; 146–153, doi:10.1109/CCGRID.2003.1199363.
17. Ferreira da Silva R, Glatard T, Desprez F. Workflow fairness control on online and non-clairvoyant distributed computing platforms. *Euro-Par 2013 Parallel Processing*, *Lecture Notes in Computer Science*, vol. 8097, Wolf F, Mohr B, Mey D (eds.). Springer Berlin Heidelberg, 2013; 102–113.
18. Ferreira da Silva R, Glatard T, Desprez F. On-line, non-clairvoyant optimization of workflow activity granularity on grids. *Euro-Par 2013 Parallel Processing*, *Lecture Notes in Computer Science*, vol. 8097, Wolf F, Mohr B, Mey D (eds.). Springer Berlin Heidelberg, 2013; 255–266.
19. Glatard T, Lartizien C, Gibaud B, Ferreira da Silva R, Forestier G, Cervenansky F, Alessandrini M, Benoit-Cattin H, Bernard O, Camarasu-Pop S, *et al.*. A virtual imaging platform for multi-modality medical image simulation. *Medical Imaging, IEEE Transactions on* jan 2013; **32**(1):110 –118, doi:10.1109/TMI.2012.2220154.
20. N’Takpe T, Suter F. Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IPDPS ’09, 2009; 1–8.
21. Hsu CC, Huang KC, Wang FJ. Online scheduling of workflow applications in grid environments. *Future Generation Computer Systems* 2011; **27**(6):860 – 870.
22. Sommerfeld D, Richter H. Efficient Grid Workflow Scheduling Using a Two-Tier Approach. *Proceedings of HealthGrid 2011*, Bristol, UK, 2011.
23. Hirales-Carbajal A, Tchernykh A, Yahyapour R, González-García JL, Röblitz T, Ramírez-Alcaraz JM. Multiple workflow scheduling strategies with user run time estimates on a grid. *Journal of Grid Computing* 2012; **10**:325–346, doi:10.1007/s10723-012-9215-6.
24. Arabnejad H, Barbosa J. Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, 2012; 633 –639.

25. Sabin G, Kochhar G, Sadayappan P. Job fairness in non-preemptive job scheduling. *Proceedings of the 2004 International Conference on Parallel Processing*, ICPP '04, 2004; 186–194.
26. Skowron P, Rzadca K. Non-monetary fair scheduling: A cooperative game theory approach. *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, ACM: New York, NY, USA, 2013; 288–297, doi:10.1145/2486159.2486169.
27. Montagnat J, Isnard B, Glatard T, Maheshwari K, Fornarino MB. A data-driven workflow language for grids based on array programming principles. *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, ACM: New York, NY, USA, 2009; 7:1–7:10, doi:10.1145/1645164.1645171.
28. Ferreira da Silva R, Glatard T. A science-gateway workload archive to study pilot jobs, user activity, bag of tasks, task sub-steps, and workflow executions. *Euro-Par 2012: Parallel Processing Workshops (CGWS-2012)*, *Lecture Notes in Computer Science*, vol. 7640, Caragiannis I, Alexander M, Badia R, Cannataro M, Costan A, Danelutto M, Desprez F, Krammer B, Sahuquillo J, Scott S, *et al.* (eds.). Springer Berlin Heidelberg, 2013; 79–88.
29. Glatard T, Montagnat J, Lingrand D, Pennec X. Flexible and Efficient Workflow Deployment of Data-Intensive Applications on Grids with MOTEUR. *International Journal of High Performance Computing Applications (IJHPCA)* Aug 2008; **22**(3):347–360.
30. Tsaregorodtsev A, Brook N, Ramo AC, Charpentier P, Closier J, Cowan G, Diaz RG, Lanciotti E, Mathe Z, Nandakumar R, *et al.*. DIRAC3. The New Generation of the LHCb Grid Software. *Journal of Physics: Conference Series* 2009; **219**(6):062 029.
31. Ferreira da Silva R. A science-gateway for workflow executions: online and non-clairvoyant self-healing of workflow executions on grids. PhD Thesis, Institut National des Sciences Appliquées de Lyon 2013.
32. Ferreira da Silva R, Glatard T, Desprez F. Self-healing of workflow activity incidents on distributed computing infrastructures. *Future Generation Computer Systems* 2013; **29**(8):2284 – 2294.
33. Jan S, Benoit D, Becheva E, Carlier T, Cassol F, Descourt P, Frisson T, Grevillot L, Guigues L, Maigne L, *et al.*. Gate v6: a major enhancement of the gate simulation platform enabling modelling of ct and radiotherapy. *Physics in medicine and biology* 2011; **56**(4):881–901.
34. Cao F, Commowick O, Bannier E, Ferré JC, Edan G, Barillot C. MRI estimation of T1 relaxation time using a constrained optimization algorithm. *MBIA'12 Proceedings of the Second international conference on Multimodal Brain Image Analysis*, *Lecture Notes in Computer Science*, vol. 7509, Yap PT, Liu T, Shen D, Westin CF, Shen L (eds.), Springer Berlin Heidelberg: Berlin, Heidelberg, 2012; 203–214.
35. Jensen J, Svendsen N. Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on* march 1992; **39**(2):262 – 267.
36. Reilhac A, Batan G, Michel C, Grova C, Tohka J, Collins D, Costes N, Evans A. Pet-sorteo: validation and development of database of simulated pet volumes. *Nuclear Science, IEEE Transactions on* oct 2005; **52**(5):1321 – 1328.
37. Henry GJ. The unix system: The fair share scheduler. *AT&T Bell Laboratories Technical Journal* 1984; **63**(8):1845–1857, doi:10.1002/j.1538-7305.1984.tb00068.x.