On-line, non-clairvoyant optimization of workflow activity granularity on grids

Rafael Ferreira da Silva, Tristan Glatard, Frédéric Desprez

On-line, non-clairvoyant optimization of workflow activity granularity on grids

Rafael Ferreira da Silva[1], Tristan Glatard[1] and Frédéric Desprez[2]

[1] University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France
{rafael.silva,glatard}@creatis.insa-lyon.fr
[2] INRIA, University of Lyon, LIP, ENS Lyon, Lyon, France
Frederic.Desprez@inria.fr

Abstract. Controlling the granularity of workflow activities executed on widely distributed computing platforms such as grids is required to reduce the impact of task queuing and data transfer time. Most existing granularity control approaches assume extensive knowledge about the applications and resources (e.g. task duration on each resource), and that both the workload and available resources do not change over time. We propose a granularity control algorithm for platforms where such clairvoyant and offline conditions are not realistic. Our method groups tasks when the fineness degree of the application, which takes into account the ratio of shared data and the queuing/round-trip time ratio, becomes higher than a threshold determined from execution traces. The algorithm also de-groups task groups when new resources arrive. The application's behavior is constantly monitored so that the characteristics useful for the optimization are progressively discovered. Experimental results, obtained with 3 workflow activities deployed on the European Grid Infrastructure, show that (i) the grouping process yields speed-ups of about 2.5 when the amount of available resources is constant and that (ii) the use of de-grouping yields speed-ups of 2 when resources progressively appear.

1 Introduction

Software-as-a-service (SaaS) platforms deployed on production grids, for instance the Virtual Imaging Platform (VIP [1]) and other science gateways [2,3,4], usually have no *a-priori* model of the execution time of their applications because (i) task costs depend on input data with no explicit model, and (ii) characteristics of the available resources, in particular network and RAM, depend on background load. Modeling application execution time in these conditions requires cumbersome experiments which cannot be conducted for every new application in the platform. As a consequence, such SaaS platforms operate in *non-clairvoyant* conditions, where little is known about executions before they actually happen. Such platforms also run in *online* conditions, i.e. users may launch or cancel applications at any time and resources may appear or disappear at any time too. Our ultimate goal is to control the behavior of these non-clairvoyant, online platforms to limit human intervention required for their operation. In other

works, we address error recovery [5] and fairness of resource allocation [6] among executions. This paper focuses on task granularity optimization.

The low performance of *lightweight* (a.k.a. *fine-grained*) tasks is a common problem on widely distributed platforms where the communication overhead and queuing time are high, such as grid systems. To address this issue, fine-grained tasks are commonly grouped into *coarse-grained* tasks [7,8,9,10,11], which reduces the cost of data transfers when grouped tasks share input data [7] and saves queuing time when resources are limited [8]. However, task grouping also limits parallelism and therefore should be used sparingly.

We consider such a granularity problem in a SaaS platform executing workflows on a grid. Workflows are compositions of *activities* that consist only of a program description. At runtime, activities receive data and spawn tasks for which the executable name and input data are known, but the computational cost and produced data volume are not. We propose an algorithm to optimize the granularity of workflow activities on non-clairvoyant online grid platforms. Our algorithm progressively discovers the characteristics of the running applications to compute a metric quantifying the fineness degree of a task group. This fineness metric includes measured task queuing times, and median-based estimations of task running times and transfer time of shared input data. Tasks are grouped when the fineness metric goes beyond a threshold learned from platform traces. In addition, a de-grouping mechanism is triggered when parallelism losses are detected, i.e. when the number of queued tasks is lower than the number of running tasks. The method is implemented in VIP, and evaluated with different applications, in production conditions, on the European Grid Infrastructure (EGI[3]). The contributions of this work are the following:

- We propose a new metric to quantify workflow activity fineness in online and non-clairvoyant conditions;
- We design task grouping and de-grouping algorithms that are triggered by the fineness metric in the control loop described in [5];
- We show, on 3 different applications, that the method provides significant speed-up in production conditions, on the European Grid Infrastructure.

To the best of our knowledge, this algorithm is the first example of task granularity control in a non-clairvoyant online context. The next Section gives an overview of the related work, Section 3 details the granularity control process, Section 4 reports experiments and results, and the paper closes with a discussion and conclusions.

2 Related Work

Muthuvelu et al. [9] 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

[3] http://www.egi.eu

tasks are grouped up to the resource capacity. This process continues until all tasks are grouped and assigned to resources. Then, Keat et al. [10] and Ang et al. [11] 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. [12] extended [9] to determine task granularity based on QoS requirements, task file size, estimated task CPU time, and resource constraints. Meanwhile, Liu & Liao [13] 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. [14] 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 capability of the resources. Zomaya and Chan [15] 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. [16,7] 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. Although this is an online approach, the solution is still clairvoyant.

3 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 section describes how η_f, η_c, τ_f and τ_c are computed, 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
```

3.1 Fineness control

Fineness degree η_f. Let n be the number of waiting tasks in a workflow activity, and m the number of task groups. Tasks related to an activity are assumed independent, but with similar execution times (bag of tasks). This hypothesis is critical for our method. 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: $T_i \bigcap_{i \neq j} T_j = \emptyset$ and $\sum_{i=1}^{m} n_i = n$. The activity fineness degree η_f is the maximum of all group fineness degrees f_i:

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

All η_f are in [0,1], and 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:

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

where 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. Yet, it assumes that all tasks in an activity have similar durations.

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:

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

Group queuing time is the max of all task queuing times in the group; 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.

Threshold value 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.

We use traces collected from VIP [1] between January 2011 and April 2012, made available through the science-gateway workload archive [17]. The data set contains 680,988 tasks (including resubmissions and replications) linked to activities of 2,941 workflows executed by 112 users; task average waiting time is about 36 min. Applications deployed in VIP are described as workflows executed using the MOTEUR workflow engine [18]. Resource provisioning and task scheduling are provided by DIRAC [19] 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)[4] which has access to some 150 computing sites world-wide and to 120 storage sites providing approximately 4 PB of storage. Fig. 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. Fig. 1 (right) shows the histogram of these values. The histogram appears bimodal, which indicates that η_f separates platform configurations in two distinct groups. We assume that these groups correspond to "acceptable fineness" (lowest mode) and "too fine granularity" (highest mode), and thus we choose $\tau_f = 0.55$. For $\eta_f \geq 0.55$, task grouping will therefore be triggered.

[4] http://www.egi.eu

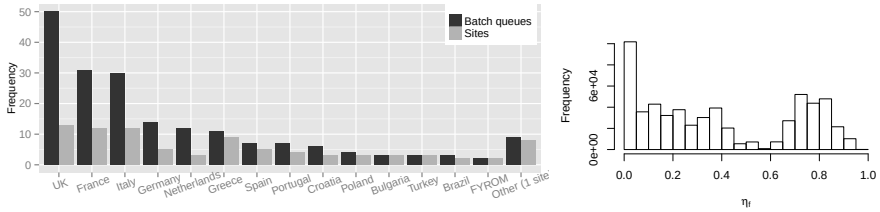

Fig. 1. Distribution of sites and batch queues per country in the biomed VO (January 2013) (*left*) and histogram of fineness incident degree sampled in bins of 0.05 (*right*).

Task grouping. We assume that running tasks cannot be pre-empted, i.e. only waiting tasks can be grouped. Algorithm 2 describes our task grouping. 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 in the next sub-section.

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
```

3.2 Coarseness control

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:

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

Let's consider a workflow 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 $Q = 6$ with the following values for waiting task groups:

i	$\max_{j\in[1,n_i]} q_j$	d_i	r_i	f_i
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

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 $Q \leq R$:

i	$\max_{j\in[1,n_i]} q_j$	d_i	r_i	f_i
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.

Let's consider that at time $t' > t$, group 11 starts running, thus $Q = 2 < R = 3$.
Eq. 3 gives $\eta_c = 0.6$. As $\eta_c > \tau_c = 0.5$, the activity is consider too coarse and task de-grouping is triggered. Then, group 13 is de-grouped to balance η_c.

Table 1. Example

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. Table 1 illustrates the method on a simple example.

4 Experiments and Results

The experiments presented hereafter evaluate the fineness control process under stationary load, and the interest of controlling coarseness under non-stationary load in a production environment.

4.1 Experiment Conditions

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.

The target computing platform for these experiments is the biomed VO where the traces used to determine τ_f 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 to the biomed VO of EGI using the DIRAC scheduler.

Three workflow activities, implementing different kinds of medical image simulation, are used in the experiments. `SimuBloch` [20] is a very short activity made of 25 concurrent tasks; task CPU time is of a few seconds; input data size is about 15 MB and output is less than 5 MB; $\tilde{t}_{\_shared}$ is about 90% of the execution time. `FIELD-II` [21] consists of 122 data-intensive concurrent tasks ranging from a few seconds to 15 minutes of CPU time (tasks have the same cost, but their duration is resource-dependent); it transfers 208 MB of input data and outputs about 40 KB of data; $\tilde{t}_{\_shared}$ ranges from 40% to 60% of the execution time. `PET-Sorteo/emission` [22] has 80 tasks of 2 CPU minutes; input data size is about 20 MB and output is about 50 MB; $\tilde{t}_{\_shared}$ ranges from 50% to 80% of the execution time.

Two sets of experiments are conducted, under different load patterns. Experiment 1 evaluates the fineness control process only 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 3 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.

For both experiments, control and tested executions are launched simultaneously to ensure similar grid conditions. 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 resubmitted too to ensure equal race conditions for resource allocation, and that each execution faces the same re-submission overhead. Five repetitions of each experiment are performed, along a time period of 4 weeks to cover different grid conditions. We use MOTEUR 0.9.21, configured to resubmit failed tasks up to 5 times, and with the task replication mechanism described in [5] activated. We use the DIRAC v6r6p2 instance provided by France-Grilles[5]. Results could not be compared to other grouping/de-grouping methods due to the lack of non-clairvoyant, online method available in the literature (see Section 2).

4.2 Results and Discussion

Experiment 1: Fig. 2 shows the makespan of `SimuBloch`, `FIELD-II`, and `PET-Sorteo/emission` executions. `Fineness` yields a significant makespan reduction for all repetitions. Table 2 shows the makespan (M) values and the number of task groups. The task

[5] https://dirac.france-grilles.fr

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.

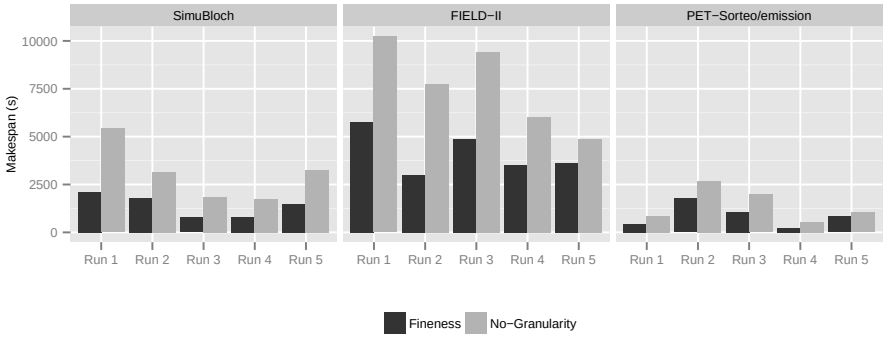

Fig. 2. Experiment 1: makespan for `Fineness` and `No-Granularity` executions for the 3 workflow activities under stationary load.

Experiment 2: Fig. 3 shows the makespan (top) and evolution of task groups (bottom). Makespan values are reported in Table 3. 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 3 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 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

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

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

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.

5 Conclusion

We presented a method to address task granularity in distributed workflows in an online and non-clairvoyant environment. We defined a metric η_f for online determination of task fineness based on queue waiting time and estimated data transfer time of shared input data. For high η_f values, tasks are considered too fine and task grouping is triggered. Queued tasks are grouped pairwise as long as the number of queued tasks is greater than the number of running tasks and η_f is considered too fine. We also define a metric η_c for online determination of task coarseness based on the ratio of the number of queued tasks related to the number of running tasks. This metric aims at maximizing resource exploitation by de-grouping tasks groups when the number of available resources increases.

The task granularity control strategy was implemented in the MOTEUR workflow engine and deployed on EGI with the DIRAC resource manager. We tested it on three applications extracted from the Virtual Imaging Platform, a science gateway for medical simulation. Two experiments were conducted, to evaluate the fineness control process only under stationary load and the fineness and coarseness control process under non-stationary load. Results showed that under stationary load, our fineness control process significantly reduces the

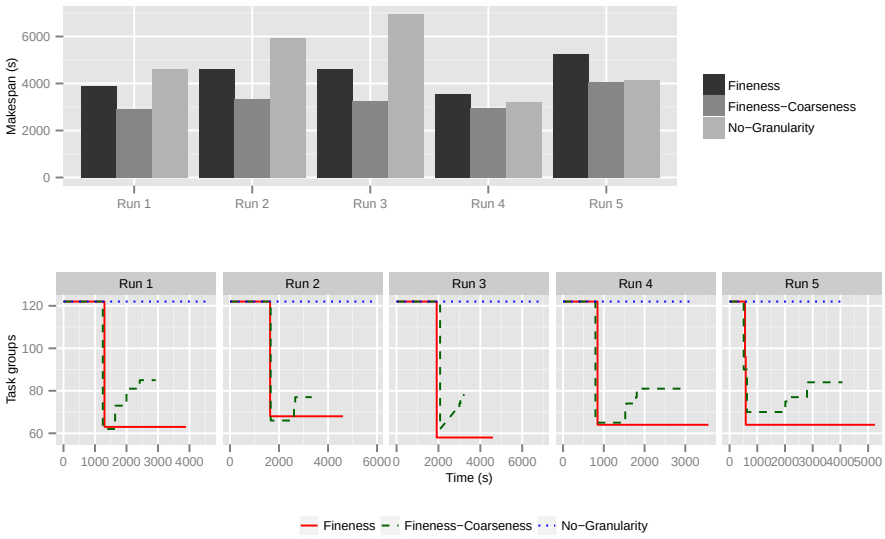

Fig. 3. Experiment 2: makespan (top) and evolution of task groups (bottom) for `FIELD-II` executions under non-stationary load (resources arrive during the experiment).

makespan of all applications. Under non-stationary load, task grouping is penalized by its lack of adaptation, but our de-grouping algorithm corrects it in case variations in the number of available resources are not too fast. In our future work, task pre-emption will be added to the method to further improve the handling of resource dynamicity. A comparative study against pilot job approaches and clairvoyant methods will also be considered. We will also study the impact of task duration variability on the proposed method.

6 Acknowledgment

This work is 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. Results obtained in this paper were computed on the biomed virtual organization of the European Grid Infrastructure (http://www.egi.eu). 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.

References

1. Glatard, T., et al.: A virtual imaging platform for multi-modality medical image simulation. IEEE Transactions on Medical Imaging **32** (2013) 110–118

2. Shahand, S., et al.: Front-ends to Biomedical Data Analysis on Grids. In: Proceedings of HealthGrid 2011, Bristol, UK (june 2011)
3. Kacsuk, P.: P-GRADE Portal Family for Grid Infrastructures. Concurrency and Computation: Practice and Experience **23**(3) (2011) 235–245
4. Barbera, R., et al.: Supporting e-science applications on e-infrastructures: Some use cases from latin america. In: Grid Computing. (2011) 33–55
5. Ferreira da Silva, R., Glatard, T., Desprez, F.: Self-healing of operational workflow incidents on distributed computing infrastructures. CCGrid'12 (2012) 318–325
6. Ferreira da Silva, R., Glatard, T., Desprez, F.: Workflow fairness control on online and non-clairvoyant distributed computing platforms. Euro-Par 2013, to appear (2013)
7. Muthuvelu, N., et al.: Task granularity policies for deploying bag-of-task applications on global grids. FGCS **29**(1) (2012) 170 – 181
8. Singh, G., et al.: Workflow task clustering for best effort systems with pegasus. In: Mardi Gras'08, New York, NY, USA, ACM (2008) 9:1–9:8
9. Muthuvelu, N., et al.: A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In: Proceedings of the 2005 Australasian workshop on Grid computing and e-research - Volume 44. ACSW Frontiers '05, Australian Computer Society, Inc. (2005) 41–48
10. Keat, N.W., et al.: Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing. Malaysian Journal of Computer Science **19** (2006)
11. Ang, T., et al.: A bandwidth-aware job grouping-based scheduling on grid environment. Information Technology Journal **8** (2009) 372–377
12. Muthuvelu, N., Chai, I., Eswaran, C.: An adaptive and parameterized job grouping algorithm for scheduling grid jobs. In: Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on. Volume 2. (2008) 975 –980
13. Liu, Q., Liao, Y.: Grouping-based fine-grained job scheduling in grid computing. In: ETCS '09. Volume 1. (2009) 556 –559
14. Soni, V.K., et al.: Grouping-based job scheduling model in grid computing. World Academy of Science, Engineering and Technology **41** (2010) 781–784
15. Zomaya, A., Chan, G.: Efficient clustering for parallel tasks execution in distributed systems. In: 18th IPDPS. (2004) 167–174
16. Muthuvelu, N., Chai, I., Chikkannan, E., Buyya, R.: On-line task granularity adaptation for dynamic grid applications. In: Algorithms and Architectures for Parallel Processing. Volume 6081 of LNCS. Springer (2010) 266–277
17. 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. In: CoreGRID'12, Rhodes, GR (2012)
18. Glatard, T., Montagnat, J., Lingrand, D., Pennec, X.: Flexible and Efficient Workflow Deployment of Data-Intensive Applications on Grids with MOTEUR. IJHPCA **22**(3) (August 2008) 347–360
19. Tsaregorodtsev, A., et al.: DIRAC3. The New Generation of the LHCb Grid Software. Journal of Physics: Conference Series **219**(6) (2009) 062029
20. Cao, F., et al.: MRI estimation of T1 relaxation time using a constrained optimization algorithm. In: Multimodal Brain Image Analysis. Volume 7509 of Lecture Notes in Computer Science. (2012) 203–214
21. Jensen, J., Svendsen, N.: Calculation of pressure fields from arbitrarily shaped, apodized and excited ultrasound transducers. IEEE T-UFFC **39**(2) (1992) 262–267
22. Reilhac, A., et al.: PET-SORTEO: Validation and Development of Database of Simulated PET Volumes. IEEE Trans. on Nuclear Science **52** (2005) 1321–1328