# Bicriteria Service Scheduling with Dynamic Instantiation for Workflow Execution on Grids

Luiz F. Bittencourt, Carlos R. Senna, and Edmundo R. M. Madeira

Institute of Computing - University of Campinas - UNICAMP
P.O. 6176, Campinas - São Paulo - Brazil
{bit,crsenna,edmundo}@ic.unicamp.br *

**Abstract.** Nowadays the grid is turning into a service-oriented environment. In this context, there exist solutions to the execution of workflows and most of them are web-service based. Additionally, services are considered to exist on a fixed host, limiting the resource alternatives when scheduling the workflow tasks. In this paper we address the problem of dynamic instantiation of grid services to schedule workflow applications. We propose an algorithm to select the best resources available to execute each task of the workflow on the already instantiated services or on services dynamically instantiated when necessary. The algorithm relies on the existence of a grid infrastructure which could provide dynamic service instantiation. Simulation results show that the scheduling algorithm associated with the dynamic service instantiation can bring more efficient workflow execution on the grid.

## 1 Introduction

A computational grid is a computing environment where heterogeneous resources are located on different administrative domains. The Open Grid Service Architecture (OGSA) [1] has moved grids to service-oriented computing (SOC) [2] based on Internet standards. Nevertheless, most grid solutions focus on the execution of task-based workflows [3], and only a few service-oriented grid environments provide dynamic service instantiation [4, 5].

In a task-oriented grid environment, the workflow tasks can potentially be executed on any available resource. This led to the development of scheduling heuristics to distribute tasks over heterogeneous computing platforms [6, 7]. On the other hand, in a service-oriented grid environment, the workflow execution is strongly dependent on the set of resources where services are already deployed. This can lead to overload on such resources, limiting the use of the grid resources according to the frequency of submissions of workflows using one or other type of service. Thus, powerful resources can be less used than middle-power ones, resulting in higher workflow execution times. To overcome this issue, we propose an algorithm to schedule services according to the incoming workflows.

* The authors would like to thank CAPES, FAPESP (05/59706-3) and CNPq (472810/2006-5 and 142574/2007-4) for the financial support.

The algorithm relies on the existence of an infrastructure which provides both the execution of workflows composed of service-tasks and the dynamic instantiation of grid services [8]. The proposed algorithm considers service instantiation costs when selecting the resources to execute tasks of the submitted workflow. Therefore, we focus on the bicriteria scheduling problem of minimizing both the number of created services and the workflow execution time (makespan).

This paper is organized as follows. Section 2 shows a background on scheduling and defines the problem. Section 3 presents the proposed algorithm, while Section 4 shows the simulation results. Related works are overviewed in Section 5. Section 6 concludes the paper and presents a couple of future directions.

## 2 Service Scheduling

The task scheduling problem is NP-Complete in general [9]. In the traditional one-criterion scheduling problem, usually the goal is to minimize tasks execution time, consequently minimizing the overall workflow execution time (makespan). In the multicriteria scheduling the objective is to minimize (or maximize) more than one criterion at the same time, generally considering the execution time as the most important criterion [10]. With the emergence of utility computing, economic costs are also being considered as an optimization criterion [11].

In the multicriteria optimization problem the optimality definition depends on how we compare two possible solutions. Actually, more than one optimal solution can exist considering that two solutions cannot be compared straightforwardly when one solution is better than the other in a set of criteria, but worse in another one. In this sense, when optimizing a multicriteria objective function, we want to find the *Pareto set* of solutions, which is the set composed of all non-dominated solutions. A solution is said non-dominated if there is no other solution which optimizes one criterion without worsening another one.

In this paper we address the bicriteria scheduling problem where the two criteria to be optimized are the makespan and the number of created services. We approach this problem with a heuristic algorithm which uses the *as late as possible* (ALAP) concept [12]. Usually, heuristics are simple to implement, have low complexity and execution time, and give good results. There are other ways of dealing with bicriteria optimization. One way is to optimize only one criterion, maintaining the other one between fixed thresholds [11]. Building an aggregate objective function (AOF) is another technique, where both objectives are combined in only one function to be optimized [10]. A common AOF is the weighted linear sum of the objectives: $f(obj_1, obj_2) = \alpha \times obj_1 + (1 - \alpha) \times obj_2$, $\alpha \in [0, 1]$. Another technique, among others, is the Multiobjective Optimization Evolutionary Algorithms (MOEA) [13].

### 2.1 Problem Definition

We consider a set of heterogeneous autonomous resources $\mathcal{R} = \{r_1, r_2, ..., r_k\}$, with associated processing capacities $p_{r_i} \in \mathbb{R}^+$, connected by heterogeneous network links. Each resource $r_i$ has a set of links $\mathcal{L}_i = \{l_{i,1}, l_{i,2}, ..., l_{i,m}\}$, $1 \leq m \leq k$,

where $l_{i,j} \in \mathbb{R}^+$ is the available bandwidth in the link between resources $r_i$ and $r_j$, with $l_{i,i} = \infty$. The resources are arranged in groups, where each group can be a LAN or a cluster, for instance. Each group is autonomous, and all groups are connected by heterogeneous links. Resources inside the same group are considered to have links with the same bandwidth between them. Figure 1 shows an example of a resources pool. Nodes are resources and edges are communication links. Edges thickness represents bandwidth, while nodes radium represents processing capacities and gray tones represent different operating systems.

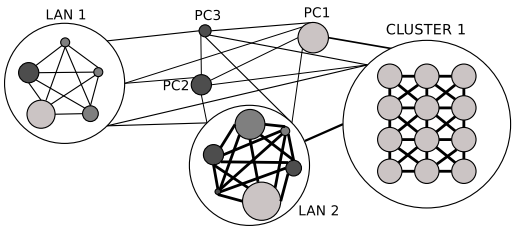


**Fig. 1.** Example of groups in the infrastructure.

Let $\mathcal{S}$ be the set of all services instantiated on all resources. Each resource $r_i$ has a set of already instantiated services $\mathcal{S}_i = \{s_{i,1}, ..., s_{i,p}\} \in \mathcal{S}$, $p \geq 0$.

A workflow is represented by a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n$ nodes (or tasks), where $t_a \in \mathcal{V}$ is a workflow task with an associated computation cost (weight) $w_{t_a} \in \mathbb{R}^+$, and $e_{a,b} \in \mathcal{E}$, $1 \leq a \leq n$, $1 \leq b \leq n$, $a \neq b$, is a dependency between $t_a$ and $t_b$ with an associated communication cost $c_{a,b} \in \mathbb{R}^+$. Each task $t_a$ has a set of candidate services $\zeta_a \subseteq \mathcal{S}, \zeta_a \neq \varnothing$, which implement task $t_a$. Therefore, a task $t_a$ can be scheduled to a resource $r_i$ *iff* $\exists\ s_{i,p} \mid s_{i,p} \in \zeta_a$.

The task scheduling is a function $schedule_{DAG} : \mathcal{V} \mapsto \mathcal{S}$. Thus, each task $t_a \in \mathcal{V}$ is mapped to a service $s \in \mathcal{S}$.

## 3 Service Scheduling Algorithm

In order to minimize the workflow's makespan, we developed a scheduling algorithm which considers the instantiation (or creation) of new services in the resources set when scheduling tasks, instead of using only existing services. This way, the workflow execution time can be minimized by creating the necessary services on resources with high processing power. But, there are some issues which must be considered when developing such an algorithm. First, the algorithm must consider how much it costs to send and deploy a service into a new host. Note that, in a utility grid, the creation of a new service may have monetary costs, thus creating too many services can lead to an expensive workflow execution. Second, neglectfully creating services can lead to resources wastefulness, such as waste of bandwidth and processing. Furthermore, this can delay the execution

of workflows already running, since processing time and bandwidth would be in constant use for transferring and creating new services abroad. Third, not creating services (or creating too less services) can lead to high workflow completion times. Thus, there is a clear trade-off between creating more services to speedup the execution, or creating less services to not waste resources (or money).

Our approach to this problem is to *create services when necessary*. This means that the algorithm will create new services only when not creating them would delay the workflow execution. To achieve this, it is mandatory to schedule tasks which are in the critical path on the best resources by creating services for them. For tasks which are not in the critical path, the algorithm uses the ALAP (as late as possible) to determine whether a service must be created. The scheduling is performed in a list-scheduling manner. At a glance, the steps of the algorithm are: select which task is the next to be scheduled; determine whether a service must be created for the selected task; and schedule the selected task on the best resource, given the service creation conditions. These three steps are repeated until all tasks are scheduled.

In the first step, the next task to be scheduled is the not scheduled one with the highest *blevel*. The *blevel* of a task $t_a$ is the length of the longest path from $t_a$ to a task with no successors (an exit node). We can assume, without loss of generality, that every workflow has one, and only one, exit task. This can be achieved by adding a costless task $t_{exit}$ and adding a set of costless edges $\{e_{a,exit} \ \forall \ t_a \in \mathcal{V} \mid t_a \text{ has no successors}\}$. On the other hand, the *tlevel* of a task $t_a$ is the length of the longest path from the entry node to $t_a$. These two attributes are known in the scheduling literature [14], and they are computed based on the costs of tasks and edges, on the capacity of the resources where they are scheduled, and on the links capacities. All these costs and capacities are previously known, as in most scheduling heuristics. For example, they can be obtained from a service repository with average costs and output sizs of the past executions, and from a resource repository, which can maintain the resources characteristics obtained from a resource monitor/discovery service.

The second step uses the ALAP of the task selected in the first step. Intuitively, the ALAP of a task is how much it can be delayed without increasing the schedule length, or by how much its *tlevel* can be increased. The ALAP of $t_a$ is computed by subtracting the *blevel* of $t_a$ from the length of the critical path. Thus, a task $t_{cp}$ in the critical path cannot be delayed, since its ALAP time is always equal to its *tlevel*. Therefore, tasks on the critical path will always be scheduled on the best available resource, and the necessary services will be created at that resource.

To determine if a service must be created to execute a task $t_a$, for each resource $r_i$ which already has the service to execute $t_a$, $t_a$ is inserted on $r_i$'s schedule, and $t_a$'s *tlevel* is computed. Let $tlevel_{t_a,r_i}$ be the *tlevel* of task $t_a$ on the schedule of resource $r_i$. If $\exists \ s_{i,p} \mid s_{i,p} \in \zeta_a, ALAP_{t_a} \leq tlevel_{t_a,r_i}$, then it is not necessary to create a new service for $t_a$, and $t_a$ is scheduled on the $r_i$ which has the smallest *tlevel* plus its execution time in that resource. Otherwise, $t_a$ is scheduled on the resource which has the smallest *tlevel* plus its execution time

plus the costs of creating the new service on the resources where it does not exist. Algorithm 1 gives an overview of these steps.

**Algorithm 1** Algorithm Overview

```
1: compute blevel, tlevel and ALAP for each task
2: while there are not scheduled tasks do
3:    t ← not scheduled task with highest blevel
4:    best_resource_t ← NULL
5:    best_time_t ← ∞
6:    R_t ← resources with services in ζ_t
7:    for all r_i ∈ R_t do
8:       calculate tlevel_t of r_i
9:       if (tlevel_t ≤ ALAP_t) AND (tlevel_t + exec_time_t < best_time_t) then
10:         best_resource_t ← r_i
11:         best_time_t ← tlevel_t + exec_time_t
12:      end if
13:   end for
14:   if best_resource_t == NULL then
15:      R ← all available resources
16:      for all r_i ∈ R do
17:         calculate tlevel_t of r_i
18:         if r_i does not have services in ζ_t then
19:            costs_create_service_t ← cost_send_code_t + cost_deploy_t
20:            tlevel_t ← tlevel_t + costs_create_service_t
21:         end if
22:         if tlevel_t + exec_time_t ≤ best_time_t then
23:            best_resource_t ← r_i
24:            best_time_t ← tlevel_t + exec_time_t
25:         end if
26:      end for
27:   end if
28:   schedule t in best_resource_t
29:   recompute tlevel and ALAP for each task
30: end while
```

The first line of Algorithm 1 computes the task attributes (*blevel, tlevel*, and ALAP). This pre-calculation is done by assuming a homogeneous virtual system with unbounded number of resources, where each resource has the best capacity available on the real system. This is done to estimate the ALAP of each task on an ideal system, which will reflect in the flexibility when searching for a resource by reducing the acceptable delay for tasks not in the critical path. After that, an iteration to schedule each task is started (line 2). The next line selects the not scheduled task with the highest *blevel* to be scheduled, while lines 4 and 5 initialize two variables used through the algorithm. The set $\mathcal{R}_t$, which contains the resources having a service able to execute task $t$, is created in line 6. After that, the iteration comprising lines 7 to 13 searches for the best resource that

can execute task $t$ without surpassing the ALAP time of $t$. If none is found (line 14), then the algorithm starts the search for the best resource in the whole set of resources (line 15). If the current resource $r_i$ does not have the necessary service (line 18), the algorithm adds to the *tlevel* of $t$ both the inherent costs of creating a new service and of executing the task (lines 19, 20). Then, the algorithm verifies if the current resource has the best *tlevel + task's execution time* of every resource already tested (line 22). If so, it is elected as the current best resource (lines 23, 24). Before the outer loop iterates, the current task is scheduled on the best resource found (line 28), and the attributes are recomputed (line 29), since the new schedule can change the *tlevel* and ALAP values.

Note that the critical path is dynamically updated on every iteration of the outer loop. If a task $t$ not in the critical path can only be scheduled on a resource which has a *tlevel* bigger than its ALAP, then $t$ will be in the critical path in the next iteration. Also, the creation of new services can be biased by introducing a multiplier to the ALAP in line 9, which would give a control over the trade-off between service creation and makespan according to the target environment.

At execution time, the creation of a new service for a task $t$ is performed after all its predecessors finish. This policy aims at not using resources before the start of the task $t$ and its workflow, since it can delay the execution of other workflows running. Additionally, the creation of services at scheduling time can waste processing time and bandwidth if a target resource leaves the grid. Besides, if we consider the situation where creating a new service or using a resource has a monetary cost, creating too many services would lead to high costs.

## 4 Experimental Results

We compared the proposed algorithm with a HEFT-like (Heterogeneous Earliest Finish Time [6]) version, differing only in that, for each task, it only considers the set of resources which have the necessary service to execute it.

**Scenarios** We varied the number of groups from 2 to 25, each with 2 to 10 resources with randomly generated capacities. Links between groups were randomly generated, as well as links between resources in the same group. For simulation purposes, links between groups never exceeded the capacities of links inside groups, since a machine inside a group cannot transmit faster than its link inside its group. Sixteen DAGs were taken for the simulations, where fifteen were randomly generated with number of nodes varying from 7 to 82, and the other one was a CSTEM DAG (Coupled Structural Thermal Electromagnetic Analysis and Tailoring of Graded Composite Structures), which weights were randomly generated but values proportional to those encountered in the original workflow [15]. All random numbers were taken from a uniform distribution.

One main parameter can influence the performance of the algorithm that does not create services: the number of services already in the resources. Let $P_{s_t,r_i} = \frac{\delta}{|\mathcal{R}|}$ be the probability of a service $s_t$, which can execute a task $t$ from the DAG, to exist in the resource $r_i$. We simulated $\delta$ varying from 1 to 5. A

simulation with $\delta = 3$, for instance, means that $\forall\ t \in \mathcal{V} : E(|\zeta_t|) = 3$, i.e., it is expected that, for each task $t$ of the DAG, the number of resources which can execute $t$ is 3. We also simulated a hypothetical situation where all services existed in all resources, thus with $\delta = |\mathcal{R}|$. This aimed at evaluating if the ALAP policy would restrict the resources usage in a manner that would increase the makespan. Another characteristic that can influence the results is the capacity of the resources where the already existing services are on. Assuming that services are generally deployed on resources with good capacity, we also simulated scenarios where the already running services could only exist on a set of resources $\mathcal{R}_{50\%} = \{r_i \in \mathcal{R} | p_{r_i} \geq median\}$ where *median* is the median of the set of processing capacities of all resources in $\mathcal{R}$. In other words, services could only exist on resources which have processing capacities higher or equal than the median of the capacities of all resources. On these scenarios, $P_{s_t,r_i} = \frac{\delta}{|\mathcal{R}_{50\%}|}$.

Each algorithm was executed 2000 times for each number of groups. On each execution a DAG was randomly chosen and costs of nodes and edges were randomly taken, both in the same interval. The costs of transferring and deploying a new service were randomly generated from a normal distribution according to the average size (250Kb) and standard deviation (120Kb) of the *.gar* files measured in our laboratory and file sizes found in the literature [16]. These files are called *grid archives* and each file has the source codes to deploy a service in the well known Globus Toolkit 4. The results show a confidence interval of 99%.

We compared the average makespan, average speedup and average schedule length ratio (SLR). The speedup means how many times faster is the achieved makespan when compared to the schedule of all tasks sequentially on the best resource (higher is better). The SLR means how many times larger is the makespan when compared to the execution of the critical path on the best resource (less is better). We also measured what was the percentage of services used by the proposed algorithm which were already existing services, represented by bars in the graphics (right axis). In the graphics, $\delta$ is the expected number of existing services for each task of the workflow. Labels *Exist* are for the algorithm which does not create services (the HEFT-like approach), labels *Prop.* are for the proposed algorithm, and labels with *50%* are for simulations where existing services could exist only on the 50% best resources.

**Schedule Length Ratio** Figure 2 shows the average SLR for $\delta = 1$. Considering that existing services could be on all resources, the proposed algorithm outperforms the HEFT-like approach by around 43%, with 2 groups, and by around 53% when there are 25 groups. This is achieved with the proposed algorithm using existing services for 34% of the tasks for 2 groups and around 21% for 10 or more groups. When considering that existing services could be only on the 50% best resources, for 2 groups the SLR is decreased by around 34%, while for 25 groups the decrease is around 49%, respectively using existing services for 39% and around 21% of the tasks. Therefore, using the ALAP to decide when create new services improves the scheduling quality and allows the workflow to use many existing services, while creating the necessary ones.

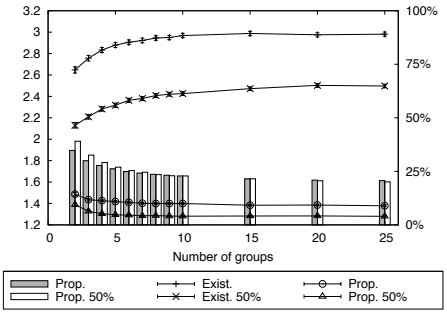


**Fig. 2.** Average SLR for $\delta = 1$.

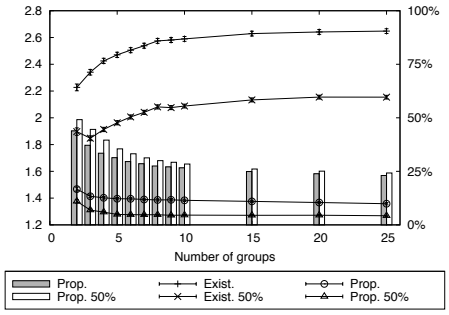


**Fig. 3.** Average SLR for $\delta = 2$.

We can observe the same pattern for $\delta = 2$ in the SLR results (Fig. 3), as well as in the speedup (Fig. 6) and makespan (Fig. 9) results, with a slightly better performance than for $\delta = 1$ when not creating new services. This is because the scheduler had more options for each task, improving the probability that a good resource would be chosen. When existing services were on any resource, the SLR for the proposed algorithm is 34% lower for 2 groups and 48% lower for 25 groups. This improvement was achieved using existing services for 44% and 23% of the tasks, respectively. When the existing services were only on the 50% best resources, the improvement varied from 27% for 2 groups, to 41% for 25 groups, using existing services for 49% and 24% of the tasks, respectively.

When $\delta = |R|$ we can observe that the proposed algorithm is slightly worse than HEFT[1]. The SLR difference ranges from 3% to 5% (Fig. 4). Note that this is a hypotetical situation, and having all services on all resources could lead to expensive workflow execution and resource squandering.

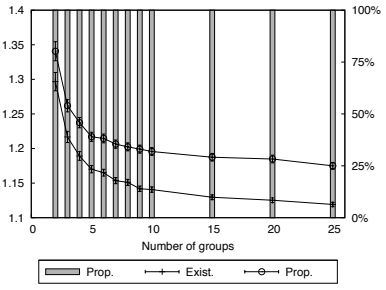


**Fig. 4.** Average SLR for $\delta = |\mathcal{R}|$.

[1] In this case the tasks could be scheduled on any resource, thus the algorithm used as comparison is the HEFT algorithm.

**Speedup** For the average speedup with $\delta = 1$ (Fig. 5), the improvement is in a range from 74%, for 2 groups, to 113%, for 25 groups, with existing services on any resource. In the scenario where existing services could be only on the 50% best resources, these numbers are 50% and 92%, respectively. The speedup for $\delta = 2$ (Fig. 6) was improved by 48% for 2 groups and by 91% for 25 groups for existing services on any resource. These numbers are 32% and 68%, respectively, when the existing services were only on the 50% best resources. When $\delta = |R|$ (Fig. 7) the proposed algorithm is slightly worse than HEFT, and the difference between them ranges from 3% to 6%. Note that the higher the number of groups, the higher the difference between the proposed algorithm and the HEFT-like one. This is also observed in the SLR results, and it can be explained by the fact that the higher the number of resources, the higher the probability of good resources being not used by the workflow. Therefore, instantiating new services can use these resources, improving the workflow execution.

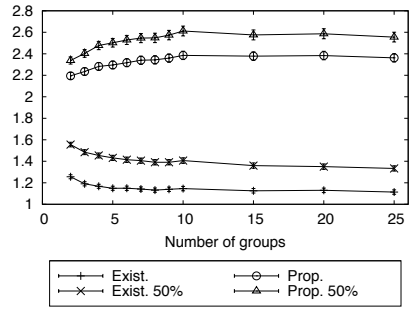


**Fig. 5.** Speedup for $\delta = 1$.

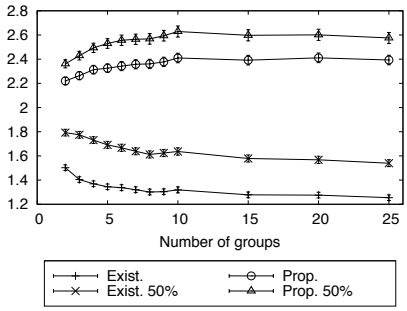


**Fig. 6.** Speedup for $\delta = 2$.

**Makespan** Figure 8 shows the average makespan for $\delta = 1$. The average for the proposed algorithm was 44% lower for 2 groups and 55% lower for 25 groups in executions with existing services on any resource. For existing services only on the 50% best resources, the improvement was of 35% for 2 groups and 49% for 25 groups. The average makespan for $\delta = 2$ (Fig. 9) shows similar results. The improvement varies from 35% to 50% when existing services were on any resource, and it varies from 28% to 42% when existing services were on the 50% best resources. For $\delta = |R|$ we can observe that the proposed algorithm is slightly worse than HEFT, with the difference ranging from 3% to 6%.

The results for $\delta = 1$ and $\delta = 2$ show that the proposed algorithm can significantly improve the performance of the workflow scheduling on service-based grids. Simulations for $\delta$ varying from 3 to 5 suggests that the HEFT-like algorithm gives better results as higher is the $\delta$, however even for $\delta = 5$ the results of the proposed algorithm are still better than the results for the algorithm which does not create new services.

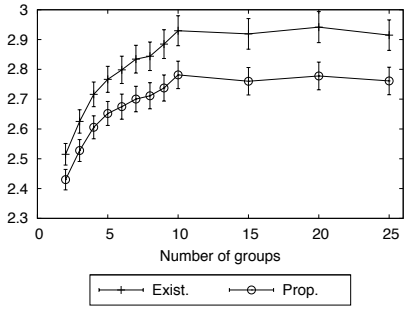


**Fig. 7.** Speedup for $\delta = |R|$.

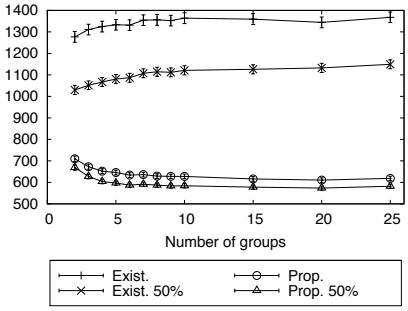


**Fig. 8.** Makespan for $\delta = 1$.

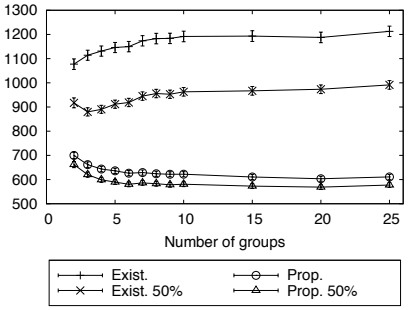


**Fig. 9.** Makespan for $\delta = 2$.

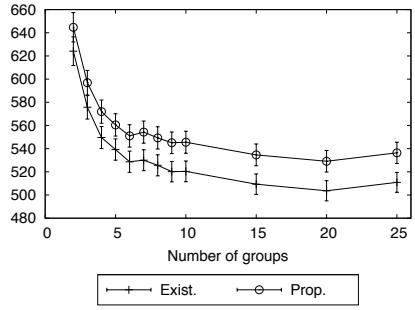


**Fig. 10.** Makespan for $\delta = |R|$.

**Remarks** The presented results show that the proposed algorithm can provide faster workflow execution by scheduling new services on resources with better performance. This is achieved by using many already existing services through the ALAP concept, determining which tasks need a new service to not delay the workflow execution. When compared to the usual execution of tasks on the existing services, the proposed strategy shows an improvement in the workflow execution by making better use of the best resources.

## 5 Related Work

While many grid middlewares focus on task-based workflows [3], others provide mechanisms for execution of workflows composed of tasks which execute on grid services [8]. The execution of workflows on grid services can be improved using dynamic service instantiation. The work described in [5] focus on mechanisms for provisioning dynamic instantiation of community services based on a highly available dynamic deployment infrastructure. However there is no mention to the necessity of a service scheduler to choose the best resources. The DynaGrid [4] is a framework for building large-scale Grid for WSRF-compliant applications that

provides mechanisms to dynamic service deployment, but there is no mention to workflow support.

In the scheduling field there are works dealing with bicriteria scheduling, but none of them addresses the problem of dynamic instantiation of services, thus, to the best of our knowledge, the problem of minimizing the number of services created and the makespan was untouched. Additionally, in [11] there is no mention to approaches to this problem.

In [10] the authors deal with the bicriteria scheduling of workflows on grids by modeling it as an extension of the multiple-choice knapsack problem, giving good results when compared with algorithms that consider makespan and budget constraints. In [17] the authors address the multicriteria optimization problem using an integer programming model focusing on price, duration, availability and success rate. Robustness and makespan are the two criteria addressed in [13], while [15] deals with the trade-off between time and reliability. Also, many works focus on economic costs [11], which can also be extended to this work if we consider monetary costs on the creation of new services.

## 6 Conclusion

In this paper we propose an algorithm to schedule workflows on service-oriented grids. The proposed bicriteria scheduling algorithm relies on the existence of dynamic service instantiation to create new services, aiming at minimizing the final execution time (makespan). Additionally, the algorithm tries to minimize the number of created services by using the ALAP (as late as possible) concept, thus creating new services only when not creating them would delay the workflow execution. The minimization of the number of created services is important to avoid wasting resources as bandwidth and processing time, as well as to avoid high budgets when considering economic costs. The shown simulations suggest that the proposed algorithm can improve the workflows speedup by up to 113% using already existing services to execute around 25% of the tasks when there are 25 groups of resources, and can improve it by 74% using existing services to execute around 35% of the tasks when there are 2 groups of resources, thus complying with the biobjective minimization aims. Also, the instantiation of new services scheduled by the proposed algorithm can maximize the resources usage while balancing the load over the available resources when there are many requests to the same service.

As ongoing work, we are integrating the algorithm within a grid infrastructure which supports workflow orchestration [8]. This infrastructure can handle execution failures by raising exceptions. Thus, as future work, we plan to develop a task and service rescheduling algorithm integrated with the exception handling mechanism.

## References

1. Forum, G.G.: Open grid service architecture, version 1.0. http://www.gridforum.org/documents/gwd-i-e/gfd-i.030.pdf (2002)

2. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Computing **9**(1) (2005) 75–81
3. Dasgupta, G.B., Viswanathan, B.: Inform: integrated flow orchestration and metascheduling for managed grid systems. In: Middleware '07: Proceedings of the 8th ACM/IFIP/USENIX international conference on Middleware, Newport Beach, California, USA (2007) 1–20
4. Byun, E.K., Kim, J.S.: Dynagrid: A dynamic service deployment and resource migration framework for WSRF-compliant applications. Parallel Computing **33**(4-5) (2007) 328–338
5. Qi, L., Jin, H., Foster, I., Gawor, J.: Provisioning for dynamic instantiation of community services. IEEE Internet Computing **12**(2) (2008) 29–36
6. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel and Distributed Systems **13**(3) (2002) 260–274
7. Bittencourt, L.F., Madeira, E.R.M.: A performance oriented adaptive scheduler for dependent tasks on grids. Concurrency and Computation: Practice and Experience **20**(9) (2008) 1029–1049
8. Senna, C.R., Madeira, E.R.M.: A middleware for instrument and service orchestration in computational grids. In: Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID'07), Rio de Janeiro, Brazil, IEEE Computer Society (2007)
9. El-Rewini, H., Ali, H.H., Lewis, T.G.: Task scheduling in multiprocessing systems. IEEE Computer **28**(12) (1995) 27–37
10. Wieczorek, M., Podlipnig, S., Prodan, R., Fahringer, T.: Bi-criteria scheduling of scientific workflows for the grid. In: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), Lyon, France, IEEE Computer Society (May 2008) 9–16
11. Yu, J., Buyya, R.: A taxonomy of scientific workflow systems for grid computing. SIGMOD Records **34**(3) (2005) 44–49
12. Simion, B., Leordeanu, C., Pop, F., Cristea, V.: A hybrid algorithm for scheduling workflow applications in grid environments (icpdp). In: OTM Conferences. Volume 4808 of LNCS., Vilamoura, Portugal, Springer (November 2007) 1331–1348
13. Canon, L.C., Jeannot, E.: Scheduling strategies for the bicriteria optimization of the robustness and makespan. In: 11th International Workshop on Nature Inspired Distributed Computing (NIDISC 2008), Miami, Florida, USA (April 2008)
14. Yang, T., Gerasoulis, A.: Dsc: Scheduling parallel tasks on an unbounded number of processors. IEEE Trans. Parallel and Distributed Systems **5**(9) (1994) 951–967
15. Dogan, A., Özgüner, F.: Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. Computer Journal **48**(3) (2005) 300–314
16. Qi, L., Jin, H., Foster, I.T., Gawor, J.: Hand: Highly available dynamic deployment infrastructure for globus toolkit 4. In: 15th Euromicro IPDP, Naples, Italy, IEEE Computer Society (February 2007) 155–162
17. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Transactions on Software Engineering **30**(5) (2004) 311–327