Dynamic Objective and Advance Scheduling in Federated Grids*

Katia Leal[1], Eduardo Huedo[2], and Ignacio M. Llorente[2]

[1] Universidad Rey Juan Carlos
Departamento de Sistemas Telemáticos y Computación
Escuela Superior de Ciencias Experimentales y Tecnología
Tulipán SN, Mósteles, Madrid, Spain
[2] Universidad Complutense de Madrid
Departamento de Arquitectura de Computadores y Automática
Facultad de Informática 28040, Spain

Abstract. In this paper we present a dynamic mapping strategy for scheduling independent tasks in Federated Grids. This strategy is performed in two steps: first we calculate a new objective, and then we apply advance scheduling to meet the new objective. The results obtained by simulation show that the combination of these two steps reduces the makespan and increases the throughput. Thus, the mapping strategy proposed meets two of the most common objective functions of tasks scheduling problems: makespan and performance of the resources. The presented algorithm is easy to implement, unlike Genetic Algorithms is fast enough to be used in a realistic scheduling, and is efficient. In addition, the information the strategy needs can be provided by any Grid Information Service, and its does not require the deployment of complex prediction services or service level agreement: it can work in any Grid.

Keywords: Federated Grids, Planning, Scheduling, Independent Tasks.

1 Introduction

Although the scheduling problem is known since long ago [1] we can just take some ideas from previous research. This is because assumptions underlying traditional systems do not hold in Grid circumstances and produce poor Grid schedules in practice [2]. This is the reason why there is a huge ongoing research effort on grid scheduling [3,4]. However, as we will see in Section 2, it is mainly centered on Partner Grids, not in Federated ones.

In a Federated Grid [5] the different participants collaborate by sharing their resources with the whole Grid. However, they do not try to achieve the same goals

* This research was supported by Consejería de Educación de la Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through BIOGRIDNET Research Program S-0505/TIC/000101, and by Ministerio de Educación y Ciencia and through the research grant TIN2006-02806. Research work also supported by the Madrid Regional Research Council under project TIC-000285-0505.

R. Meersman and Z. Tari (Eds.): OTM 2008, Part I, LNCS 5331, pp. 711–725, 2008.
© Springer-Verlag Berlin Heidelberg 2008

as they have to satisfy their own users demands. In doing so, each participant can use his own internal resources, but also the grid resources that the rest of participant are willing to share. Each participant decides which resources to contribute with, who can use those resources, and the access policy to them. Federated Grids are explained in more detail in Section 3.

In a previous work [6], we proposed an alternative to GridWay 's current scheduling policy based on a performance model [7] that allows to parametrize and compare different Grids. In this way, we characterized the performance of a Federated Grid by means of the r_∞, and $r_{1/2}$ parameters [8] in order to determine the number of jobs to submit to each of the grid infrastructures forming the Federated Grid to maximize the throughput of the internal resources. Finally, the simulation results showed that the enhanced but *static* scheduling policy proposed maximized the throughput of internal resources, but without increasing the computational time, and provided a fair distribution of the jobs.

In Section 4, we introduce a *dynamic* mapping strategy for scheduling independent tasks in Federated Grids in two steps: first we calculate a new dynamic objective by means of the r_∞, and $r_{1/2}$ parameters, and then we apply advance scheduling to meet that objective. This dynamic mapping strategy, here after **DO-AS** (Dynamic Objective and Advance Scheduling), meets two of the most common objective functions of tasks scheduling problems [3]: makespan and performance of the resources. Also in Section 4, we present the pseudocode of DO-AS to demonstrate its simplicity of implement, and that the information it requires can be provided by any Grid Information Service. In addition, DO-AS does not require the deployment of complex prediction services, like the Network Weather Service (NWS), or Service Level Agreement (SLA): it can work in any Grid. The simulation results of Section 6 show the efficient of the dynamic mapping strategy because it provides better makespan and throughput than GridWay 's current scheduling policy. As we will summarize in Section 7, a significant difference between both strategies is that while DO-AS has been built having in mind Federated Grid restrictions, such as different types of users and resources, GridWay [9] applies scheduling strategies that are better fitted to Enterprise or Partner Grids with direct access to all resources.

2 Related Work

In this Section, we will review those approaches that for scalability problems are more suitable for local schedulers or workload managers than for scheduling in Federated Grids.

For its simplicity, Genetic Algorithm (GA) is one of the most popular mechanisms used for scheduling independent jobs in grid environment. However, the GA is too slow to be used in a realistic scheduling due to its time-consuming iteration. One of the factors which determine whether an approach is worthy of pursuit is the time required to evaluate a solution [10,11]. The Improved Genetic Algorithm (IGA) was proposed to enhanced the search performance of conventional GA [12]. However, worst experimentation for 8 resources and 60 tasks

showed time-consuming near to 1 second. Probably, the algorithm will consume much more time in case of thousands of jobs, and resources. As we will see in later Sections, the performance of DO-AS is independent of the number of jobs, and resources: the calculation of a new objective is a extremely light process. Thus, unlike GA, DO-AS is fast enough to be used in a realistic scheduling.

Since Min-min, and Max-min [13] are static algorithms, the assignment of tasks is fixed a priori: these algorithms need prediction information on processor speeds and tasks lengths to compute tasks priorities. However, cost estimate based on static information is not adaptive to situations such as one of the nodes selected to perform a computation fails, becomes isolated from the system due to network failure, or is so heavily loaded with jobs that its response time becomes longer than expected. Another approach that does not require such information was proposed to improve the previous ones, but the simulation results demonstrated that it was only the next to the best of those algorithms [14]. In contrast, DO-AS is dynamic, does not require information on processor speeds and task lengths, and the results we show have been obtained from a simulated Federated Grid based on a real testbed.

Federated Grids are heterogeneous and highly dynamic environments, on which meta-schedulers are situated on the top level of the system architecture, far from the resources. Thus, meta-schedulers have a general view of the whole Federated Grid infrastructure, not a particular one. This is why we can not pretend to apply fine-grained techniques more suitable to local schedulers or workload managers that are close to the resources, to meta-schedulers. Instead, what we need are light, decoupled, and coarse-grained techniques.

3 Federated Grids

As it can be seen in Figure 1, in a Federated Grid there are different types of resources:

- ❒ *Internal Resources*: these are the resources directly accessible by the meta-scheduler through the corresponding local workload manager. That is, the resources owned by the particular research center, laboratory or company.
- ❒ *External Resources*: these are the resources accessible through a remote metascheduler by means of a WSRF interface. We can classify Enterprise, Partner and Utility Grids [6] in this category.

Also, we have identified internal, external, and direct users. They differ in the way, and in the rights they have to access resources:

- ❒ *Internal Users*: the jobs submitted by these users through the meta-scheduler can be executed in both internal, and external resources.
- ❒ *External Users*: all the jobs received by the meta-scheduler through the Web Services Resource Framework (WRSF) interface will be from external users.
- ❒ *Direct Users*: the meta-scheduler cannot control the jobs submitted by this type of internal users. However, they are important since they have an influence in the load of the resources.

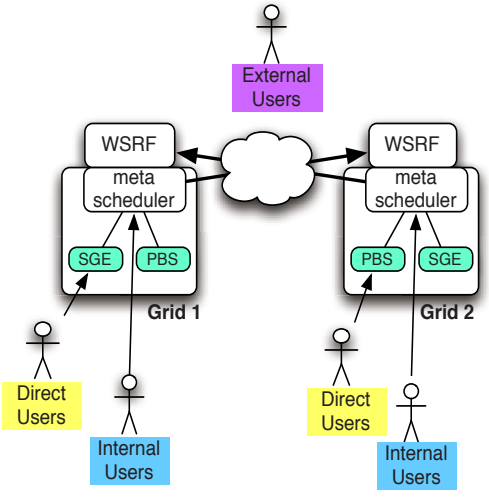

Fig. 1. Example of a Federated Grid

The GridWay meta-scheduler [9] performs job execution management and resource brokering on a grid consisting of distinct computing platforms managed by Globus services. GridWay allows unattended, reliable, and efficient execution of single, array, or complex jobs on heterogeneous and dynamic grids. A Web Service - Grid Resource Allocation and Management (WS-GRAM) Globus Toolkit service hosting a GridWay meta-scheduler acts as a grid gateway between two grid infrastructures [15]. We have named *GridGateWay* to this entity that acts as a grid gateway. The GridGateWay will be managed as a common resource in the first grid infrastructure, and will act as a proxy of its users in the second grid. The second grid infrastructure will appear in the information system on the first as a normal WS-GRAM publishing an aggregation of grid resources; and enabling submission, monitoring and control of jobs across the several grids forming the Federated Grid. Thus, we can implement the Federated Grid depicted in Figure 1 by using GridWay as the meta-scheduler, and the WS-GRAM as the WSRF interface. Other metaschedulers [4] or remote submission interfaces, like the OGF Basic Execution Service (BES), could be used as well.

4 Dynamic Mapping Strategy: DO-AS

The main objective of this work is to provide a *dynamic* mapping strategy for scheduling independent tasks in Federated Grids to reduce the makespan, and to increase the throughput of resources. The DO-AS algorithm is divided in two steps: *DO*, on which by means of the r_∞, and $r_{1/2}$ parameters a new dynamic objective is calculated, and *AS*, on which advance scheduling is applied to meet the new objective.

4.1 Dynamic Objective: DO

The *Objective* determines the number of jobs that should be submitted to each of the grid infrastructures forming the Federated Grid , in order to maximize the throughput of the internal resources, but without increasing the makespan. We have used the equation that represents the best characterization of the Federated Grid to obtain these numbers. The characterization can be obtained if we take the line that represents the average behavior of the system, as proposed by Hockney, and Jesshope [8]:

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

In the Equation 1 n represents the number of completed tasks as a function of time t. The other parameters are:

- ❒ **Asymptotic performance** r_∞: is the maximum rate of performance in tasks executed per second. In the case of an homogeneous array of N processors with an execution time per task T, we have $r_\infty = N/T$.
- ❒ **Half-performance length** $n_{1/2}$: is the number of tasks required to obtain the half of the asymptotic performance. This parameter is also a measure of the amount of parallelism in the system as seen by the application.

The linear relation represented by Equation 1 can be used to define the performance of each grid infrastructure forming the Federated Grid (tasks completed per second), then by using the *aggregation* or *federation* model [16] we can determine the Objective even in case of having more than 2 participants.

Pseudo code for DO algorithm is given in Algorithm 1, which shows how to calculate a new objective every `OBJECTIVE_INTERVAL` seconds if there are jobs to schedule (line 1), and if there are enough samples from jobs already finished (line 3) in each grid infrastructure to properly characterize the corresponding grid. The algorithm first calculates the linear relation of Equation 1 for internal resources to define its performance (line 4-7). In lines 8 to 11 does the same for the external resources. Then, in lines 12 to 13 the algorithm calculates the linear equation of the whole Federated Grid as an *aggregation* from the linear relations of each infrastructure forming the Federated Grid. If there are any problem calculating the linear relations, the algorithm will use a default objective, that consist in submitting half of the unscheduled jobs to internal resources, and the other half to external resources. Once the performance of the Federated Grid has been defined, the algorithm can determine the time `X` for the Federated Grid to complete `unscheduledJobs.size()` jobs (line 14). In lines 15 to 17 the algorithm determines the jobs that internal resources could execute in time `X`. In order to increase the throughput of internal resources, the algorithm calculates the jobs that will be sent to external resources as a function of the jobs that will be submitted to internal resources (lines 28-20). Finally, the algorithm sets the new objectives (lines 21-22), and programs a new event for the next update of the objective (line 23).

As it can be seen, Algorithm 1 only considers one internal resource, and one external resource to calculate the performance of the whole Federated Grid.

Algorithm 1: UpdateObjective()

```
1  if (unscheduledJobs.size() < 1) then
2      sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE_OBJECTIVE);
3  if ((internalX.size() >= MIN_IN_SAMPLES) && (externalX.size() >= MIN_EX_SAMPLES)) then
4      internalRegression = linearEquation(internalX, internalY);
5      if (internalRegression.getR_Inf() < 0 || internalRegression.getN_1_2() > 0) then
6          defaultPrediction();
7          sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE_OBJECTIVE);
8      externalRegression = linearEquation(externalX, externalY);
9      if (externalRegression.getR_Inf() < 0 || externalRegression.getN_1_2() > 0) then
10         defaultPrediction();
11         sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE_OBJECTIVE);
12     r_inf_FG = internalRegression.getR_Inf() + externalRegression.getR_Inf();
13     n_1_2_FG = internalRegression.absN_1_2() + externalRegression.absN_1_2();
14     X = (unscheduledJobs.size() + n_1_2_FG) / r_inf_FG;
15     newInternalObjective = (internalRegression.getR_Inf() * X) + internalRegression.getN_1_2()
16     if (newInternalObjective > unscheduledJobs.size()) then
17         newInternalObjective = unscheduledJobs.size();
18     newExternalObjective = unscheduledJobs.size() - newInternalObjective;
19     if (newExternalObjective < 0) then
20         newExternalObjective = 0;
21     internalObjective = newInternalObjective;
22     externalObjective = newExternalObjective;
23 sendInternalEvent(OBJECTIVE_INTERVAL, UPDATE_OBJECTIVE);
```

However, in case of having a more complex grid scenario with more than two participants, it would be very easy to write a generic version of Algorithm 1.

For example, at simulation time if there are 10 internal results (completed jobs on internal resources), 11 external results (completed jobs on external resources), and 319 unscheduled jobs, the DO algorithm determines that the time `X` (line 14) for the Federated Grid to complete those 319 unscheduled jobs is 37.14093118180266 minutes. Thus, internal resources could execute (line 15) 179 of those 319 unscheduled jobs in that time, and external resources (line 18) the 140 remaining jobs. As it can be seen, to determine a new objective, DO only needs to know the completion times of already finished jobs, both on internal, and external resources. For the scheduler is very easy to determine the completion time: this value is stored every time the scheduler receives a finished jobs from a resource. We can also say this algorithm is fast enough to be used in a realistic scheduling: its complexity can be reduced to calculate the value of two linear equations, one from the values of already finished jobs on internal resources, and the other for those completed jobs on external resources. All in all, this algorithm can work in any Grid.

4.2 Advance Scheduling: AS

In our previous work [6] we got throughput increase on internal resources without increasing the completion time. Now we get makespan reduction with the implementation of this advance scheduling algorithm. The main difference with the previous scheduler is that the new one does not wait for a free PE, instead the scheduler queues jobs in the target resources.

The details of the scheduler are shown in Algorithm 2. In this pseudo code, the AS algorithm maps no more than `DISPATCH_CHUNK` jobs to resources every `SCHEDULING_INTERVAL` seconds if there are unscheduled jobs (line 1). First, the algorithm determines how many of the `DISPATCH_CHUNK` jobs to schedule are internal jobs (line 3). As we said before, in a Federated Grid GridWay can receive jobs form internal, as well as from external users. Since AS has an objective to meet, it has to determine how many of the `internalJobs` jobs to schedule should be submitted to internal resources and how many to external. As you can see in line 4, `toInternal` is proportional to number of jobs that the DO algorithm predicts that has to be submitted to internal resources. For example, if the prediction says that 120 of 300 unscheduled jobs should be submitted to internal resources, only the 40% of the `internalJobs` jobs can be scheduled to internal resources. The rest of the `internalJobs` jobs are going to be schedule to external resources (line 5). Then, while `scheduledJobs` jobs keeps less than `DISPATCH_CHUNK`, and there are available internal and external resources (line 6), AS maps the next job (line 7). If the job to schedule is internal (line 8), and if there are available internal resources, and havent met the internal schedule objective (line 9), then job is scheduled to an internal resource (line 10). Basically, `scheduleToInternalResource` queues jobs to a limit of `Number of PEs()*MAX_RUNNING_RESOURCE_FACTOR`. Otherwise, if there are not available internal resources or the internal schedule objective has been met AS evaluates line 15. Thus, if there are available external resources, and havent met the external schedule objective, the job is scheduled to an external resource. `scheduleToExternalResource` (line 16) behaves like `scheduleToInternalResource`. Otherwise, if the job wasn't internal (line 20), and in order to avoid the situations on which a participant of the Federated Grid can receive from another one a job previously submitted to it, GridWay only schedules external jobs to internal resources (line 21). Finally, all the jobs that are being scheduled are dispatched (line 24), and AS programs a new event for the next schedule (line 25).

5 Design and Implementation

We have previously [16,17] set up a simple, but real infrastructure, where resources from the EGEE infrastructure are accessed through a GridGateWay. However, prior to run DO-AS on a real production infrastructure, we have first implemented it on a simulated environment. The deployment on a real environment will require involvement of a large number of active users and resources, which is very hard to coordinate and build, and would prevent repeatability of results. Thus, the simulation appears to be the easiest way to analyze the modified scheduling policy. Based on the simulation results, we can later encourage or discourage the deployment on a real production environment. We have used the well known GridSim toolkit[1] to simulate a Federated Grid.

[1] http://www.gridbus.org/gridsim/

Algorithm 2: Scheduler()

```
1  if (unscheduledJobs.size() == 0) then
2      sendInternalEvent(SCHEDULING_INTERVAL, SCHEDULE_NOW);
3  internalJobs = numInternalJobs(DISPATCH_CHUNK, unscheduledJobs);
4  toInternal = (internalJobs*internalObject)/unscheduledJobs.size();
5  toExternal = internalJobs - toInternal;
6  while ((scheduledJobs < DISPATCH_CHUNK) && (availableInternal || availableExternal)) do
7      j = (Job)it.next();
8      if (j.isInternalJob()) then
9          if (availableInternal && (scheduledToInternal < toInternal)) then
10             if ((availableInternal = scheduleToInternalResource(j)) == true) then
11                 scheduledToInternal++;
12                 scheduledJobs++;
13                 remove();
14                 continue;
15         if (availableExternal && (scheduledToExternal < toExternal)) then
16             if ((availableExternal = scheduleToExternalResource(j)) == true) then
17                 scheduledToExternal++;
18                 scheduledJobs++;
19                 remove();
20     else
21         scheduleToInternalResource(j);
22         scheduledJobs++;
23         remove();
24  dispatch();
25  sendInternalEvent(SCHEDULING_INTERVAL, SCHEDULE_NOW);
```

5.1 GridWaySim Entities

We have called *GridWaySim* to our simulation of a Federated Grid. Next, we explain the different participating entities of GridWaySim.

- **GridWay:** the *GridWay* entity represents a generic GridWay meta-scheduler implementing the DO-AS algorithm.
- **Testbed**: a *Testbed* represents a generic set of grid resources.
- **User:** the *User* models a user that submits experiments to a GridWay broker. We use this entity to represent internal as well as external users. The functionality of each user includes the submission of experiments to the correspondent broker, and waiting for it completion.
- **Experiment:** an *Experiment* is a collection of jobs. We use this entity to recover important information about the experiment (such as the start and end times), and of all its jobs.
- **Job:** the *Job* entity represents a generic job submitted to the grid. This entity provides specific information about each job: start time, end time, and CPU time among others. We can represent jobs of different computation times, and with different input and output file sizes.
- **Workload:** the Workload entity submits jobs by reading resource traces from a file. Thus, our jobs are competing with the jobs submitted by the Workload entity.
- **GridWaySim:** this entity represents the whole simulation, and is responsible of the creation of the main simulated entities: GridWay brokers, Users, Testbeds, and Workloads.

6 Experiments and Results

We have implemented two versions of GridWaySim that only differ in the mapping strategy they implement. Thus, there is a *Normal* version that implements GridWay 's current scheduling policy, and a *DO-AS* version. As a result, the mapping policy is the only factor that can cause throughput, and makespan variations between these two GridWaySim versions. Apart from that, Normal, as well as DO-AS versions relies on the same configuration, with the same number of users that submit at the same time the same experiment with the same number of jobs (each with the same length, and input and output files size) to the same broker. Also, the number of brokers and resources is the same.

6.1 Test Scenario

As depicted in Figure 2, we have used the same scenario than in our previous work. As it can be seen, in this test scenario there are only two grid resources: the DSA (Distributed System Architecture) and the LCG (LHC Computing Grid). The DSA testbed represents the resources of the Distributed System Architecture research group at the Universidad Complutense de Madrid. In the same way, the LCG testbed represents the Large Hadron Collider (LHC) Computing Grid. From the point of view of a *DSA internal user*, the DSA GridWay is her broker, the DSA resources are internal resources, and the LCG resources are external resources. In the same way, all the jobs received by the LCG GridWay through the Globus WS-GRAM interface are from external users.

Tables 1, and 2 summarize the number of computing elements, aka PEs (Processing Elements), and MIPS (Millions Instructions Per Second) of each machine in the DSA, and LCG infrastructures.

6.2 Simulation Entities and Parameters

When the simulation starts, GridWaySim creates 3 Users, 2 GridWay brokers, 1 DSATestbed, 1 LCGTestbed, and 1 Workload. Each of which is an independent thread attending petitions in their `body()` method.

Next follows the exact configuration of both versions of GridWaySim:

- **GridWay :** since we need to interconnect the DSA, and LCG grids to form a federation, we have to instantiate 2 GridWay broker, 1 for each grid infrastructure.
- **Testbed:** the resources of the DSA research group are represented y the *DSATestbed* entity, and the LCG ones by the *LCGTestbed* entity. Each follows the configurations depicted in Tables 1 and 2, respectively.
- **Experiment:** every Experiment is a collection of 550 equal Jobs.
- **Job:** the main parameters of each Job are the length or size (in Million of Instructions, MI) of the Job to be executed, the input files (in bytes), and the output files (also in bytes) to be submitted to the corresponding resource. All Jobs have the same values for the three parameters: the size is 6,000,000

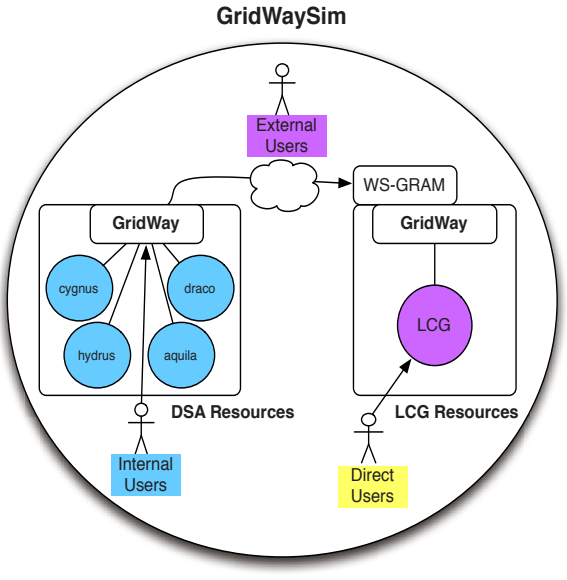

Fig. 2. A simple test scenario

Table 1. Characteristics of the machines in the DSA research testbed

Machine	PEs	MIPS/PE
hydrus	4	9787
aquila	5	9787
orion	1	9787
cygnus	2	6536
draco	1	6536

Table 2. Characteristics of the machines in the LCG research testbed

Machine	PEs	MIPS/PE
machine0	800	9787
machine1	640	6536
machine2	560	4902

MI, the input file size is 1,000,000 bytes, and the output file size is 2,000,000 bytes.

❒ **User:** when a User is created, we have to indicate a submit time for her Experiment. Each user only submits one Experiment 48 hours after the previous one. The first User submits her Experiment at 12:00 of the second day of the simulation. Thus, each User submits her experiment to the DSA GridWay at 12:00 of the corresponding day of simulation.

- **Workload:** the Workload entity submits 188,041 jobs to the LCGTestbed at the time specified in the trace file. For this reason, the LCG grid resources might not be available at certain times. The file follows a standard workload[2]. As trace file, we have used the LCG Grid Log that contains 11 days of real activity from multiple nodes that make up the LCG (Large Hadron Collider Computing Grid[3]). Next, we enumerate some details about this testbed:
  - **Number of jobs submitted:** 188,041. The log specifies the submit time and the run time of each job.
  - **Start time:** Sun Nov 20 00:00:05 GMT 2005.
  - **End time:** Mon Dec 05 10:30:24 GMT 2005.
  - **Maximum number of machines:** 170.
  - **Maximum number of computing elements:** 24,515.

  Although the number of PEs of the real LCG testbed is 24,515, we do not know the real number of PEs involved in this experiment. So, after running some simulations, we decided to reduce the number of PEs in our simulated LCG testbed to those in the Table 2. We have reduced the number of PEs in order to force LCG saturation scenarios.

Next follows detailed information about DO-AS mapping strategy parameters:

- *Default Objective*: when the simulation starts, and since there is not enough information of finished jobs for DO algorithm to calculate a new objective, we set a default objective as follows. Half of the resources should be submitted to internal resources, and the other half to external resources.
- *OBJECTIVE_INTERVAL*: the DO algorithm is called every 30 seconds.
- *MAX_RUNNING_RESOURCE_FACTOR*: the value of this factor is 3, so if a resource has 5 PEs then AS can queue a maximum of 15 jobs in that resource.
- *SCHEDULING_INTERVAL*: the AS algorithm is called every 30 seconds.
- *DISPATCH_CHUNK*: the AS algorithm schedules 15 jobs each time.

6.3 Results

Table 3 summarizes the results of both GridWaySim versions when mapping 550 jobs. We can explain these results based on the level of saturation of the LCG resource, or what is the same, knowing the number of available free PEs. Thus, each User submits his Experiment in a concrete instant of the simulation that corresponds with a different LCG saturation level. So, when User-0 submits his Experiment, there is an ideal scenario on which the LCG infrastructure always presents free PEs: *low saturation scenario*. The *medium saturation scenario* is the one suffered by User-1, in this case the LCG resource has less free PEs. Finally, User-2 coexists within a *high saturation scenario* of LCG in which there are few available PEs. Since the *Normal* algorithm only submits jobs when the

[2] http://www.cs.huji.ac.il/labs/parallel/workload/
[3] http://lcg.web.cern.ch/LCG/

Table 3. DO-AS Vs Normal simulation when scheduling 550 jobs

	DO-AS			Normal		
	DSA	LCG	Makespan	DSA	LCG	Makespan
User-0	67	483	1:30:21	26	524	1:37:26
User-1	70	480	1:37:55	113	437	2:06:40
User-2	70	480	1:38:41	472	78	7:00:41

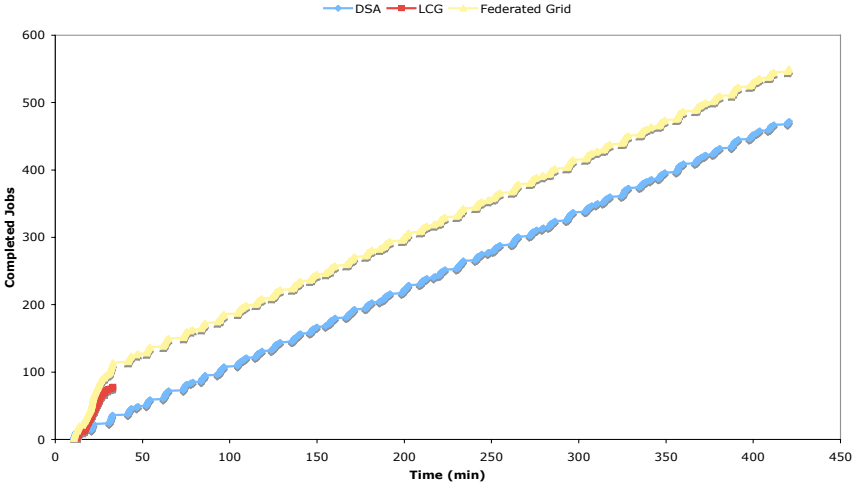

Fig. 3. Throughput achieved by using the *Normal* algorithm on DSA, LCG, and Federated Grid infrastructures for User-2 (LCG highest saturation situation)

resources have free PEs, clearly the results correspond with the different saturation scenarios: as higher saturation of LCG, more jobs are executed on internal resources, and more time is needed for the Experiment completion. In contrast, since the *DO-AS* algorithm queues jobs in advance instead of waiting for PEs availability, its performance is more uniform: the number of jobs submitted to each infrastructure, and the makespan are almost the same for the three Users. In general, User-0 compared with Users 1, and 2 obtains the best results, but this is obvious since the waiting time on LCG queues is practically null. Also, for all the Users the *DO-AS* algorithm always reduces the makespan obtained by the *Normal* algorithm. The most remarkable reduction is obtained by User-2. This spectacular reduction is due to the queuing of jobs on resources, instead of waiting for free PEs as *Normal* algorithm does. It can be also appreciated the fairness distribution of jobs for User-0 performed by *DO-AS*. While the *Normal* policy only executes 26 of 550 jobs on the DSA internal resource, *DO-AS* in less time executes 67 jobs.

In Figures 3, and 4 we can graphically see the performance difference between both algorithms for User-2, that represents LCG highest saturation situation.

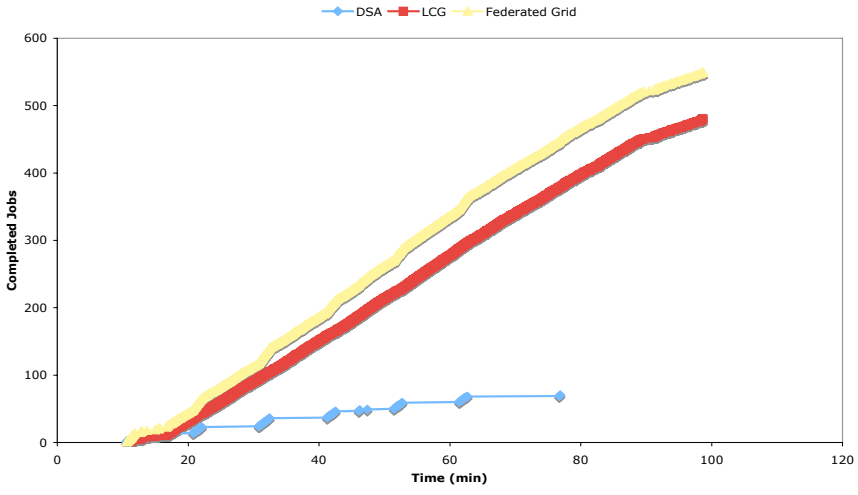

Fig. 4. Throughput achieved by using the *DO-AS* algorithm on DSA, LCG, and Federated Grid infrastructures for User-2 (LCG highest saturation situation)

7 Conclusions and Future Work

The deployment of a Federated Grid by means of the GridWay technology is feasible, however GridWay applies mapping strategies that are better fitted to isolated Enterprise or Partner Grids with direct access to all resources. In this paper, we have presented a new mapping strategy as an enhancement of a previous static algorithm. Thus, the algorithm proposed, the DO-AS, is a *dynamic* mapping strategy for scheduling independent tasks in Federated Grids. This strategy is performed in two steps: first calculating a new dynamic objective by means of the r_∞, and $r_{1/2}$ parameters, and then applying advance scheduling to meet the objective. In contrast of GA approaches, the performance of DO-AS is independent of the number of resources, and of jobs to schedule. In addition, unlike them, DO-AS is fast enough to be used in a realistic scheduling. Also, our algorithm considers the performance of the infrastructures forming the Federated Grid, not only their status. In addition, we have implemented a simulation environment to evaluate the DO-AS policy based on a simple, but real testbed. Finally, the simulation results provided by GridWaySim show that DO-AS compared with GridWay 's current scheduling policy reduces the makespan, and increases the performance of the resources in the three saturation situations presented.

As we saw in Section 4, this algorithm is very easy to implement, and the information it needs can be provided by any Grid Information Service. In addition, for the implementation of DO-AS in a scenario like the one presented in Section 6.1, we do not require the deployment of complex prediction services, like the Network Weather Service (NWS), or Service Level Agreement (SLA): DO-AS can work in any Grid.

Our current work focuses on the simulation of a more complicated testbed to deeply test the proposed algorithm. Also, we want to improve the DO-AS algorithm by adapting at runtime the values of important parameters. In addition, we want to investigate the effects of sharing policies in the proposed mapping policy. Finally, we will implement the DO-AS algorithm in GridWay for its deployment in a real infrastructure.

Acknowledgments. Thanks to the GSyC/Libresoft Research on Free Software Engineering group at Rey Juan Carlos University for the hardware support to run the simulations.

References

1. Ullman, J.D.: NP-Complete Scheduling Problems. Journal of Computer and System Sciences 10(3), 384–393 (1975)
2. Berman, F.: High-Performance Schedulers. In: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann, San Francisco (1998)
3. Dong, F., Akl, S.G.: Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report 2006-504, Ontario Queenâs University (January 2006)
4. Andrieux, A., Berry, D., Garibaldi, J., Jarvis, S., MacLaren, J., Ouelhadj, D., Snelling, D.: Open Issues in Grid Scheduling. Technical Report ISSN 1751-5971, UK e-Science Institute (October 2003)
5. de Assuncao, M.D., Buyya, R., Venugopal, S.: InterGrid: A Case for Internetworking Islands of Grids. Concurrency and Computation: Practice and Experience (CCPE) (2007)
6. Leal, K., Huedo, E., Montero, R.S., Llorente, I.M.: Scheduling Strategies in Federated Grids. In: Proceedings of the 2008 High Performance Computing & Simulation Conference (HPCS 2008) (2008)
7. Montero, R.S., Huedo, E., Llorente, I.M.: Benchmarking of High Throughput Computing Applications on Grids. Parallel Computing 32(4), 267–269 (2006)
8. Hockney, R., Jesshope, C.: Parallel Computers 2: Architecture, Programming, and Algorithms. Adam Hilger Ltd (1988)
9. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. Software – Practice and Experience 34(7), 631–651 (2004)
10. Casavant, T.L., Kuhl, J.G.: A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering 14(2), 141–154 (1988)
11. Braun, T.D., Siegel, H.J., Beck, N., Blni, L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B.: A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In: The 17th IEEE Symposium on Reliable Distributed Systems (1998)
12. Yin, H., Wu, H., Zhou, J.: An Improved Genetic Algorithm with Limited Iteration for Grid Scheduling. In: Proceedings of the Sixth International Conference on Grid and Cooperative Computing (GCC 2007) (2007)

13. Freund, R.F., Gherrity, M., Ambrosius, S., Campbelly, M., Halderman, M., Hensgenz, D., Keithy, E., Kiddz, T., Kussowy, M., Limay, J.D., Mirabilea, F., Moorex, L., Rusty, B., Siegel, H.J.: Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet. In: Proceedings of the 7th International IEEE Heterogeneous Computing Workshop (HCW 1998) (1998)
14. Fujimoto, N., Hagihara, K.: A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks. In: Proceedings of the 2004 International Symposium and the Internet Workshops (SAINTW 2004) (2004)
15. Llorente, I.M., Montero, R.S., Huedo, E., Leal, K.: A Grid Infrastructure for Utility Computing. In: Proceedings of the Third International Workshop on Emerging Technologies for Next-generation GRID (ETNGRID 2006). IEEE Computer Society Press, Los Alamitos (2006)
16. Vázquez, C., Fontán, J., Huedo, E., Montero, R.S., Llorente, I.M.: A Performance Model for Federated Grid Infrastructures. In: Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and network-based Processing (PDP 2008), pp. 188–192 (2008)
17. Vázquez, C., Huedo, E., Montero, R.S., Llorente, I.M.: Evaluation of A Utility Computing Mode based on Federation of Grid Infrastructures. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 372–381. Springer, Heidelberg (2007)