Self-adaptivity for Grid Applications. An Efficient Resources Selection Model Based on Evolutionary Computation Algorithms

María Botón-Fernández[a], Miguel A. Vega-Rodríguez[b], Francisco Prieto Castrillo[a]

[a] *Ceta-Ciemat, Dept. Science and Technology. Trujillo, Spain*
[b] *University of Extremadura, Dept. Technologies of Computers and Communications. Cáceres, Spain*

Abstract

Over the last few years, the adaptation ability has become an essential characteristic for grid applications due to the fact that it allows applications to face the dynamic and changing nature of grid systems. This adaptive capability is applied within different grid processes such as resource monitoring, resource discovery, or resource selection. In this regard, the present approach provides a self-adaptive ability to grid applications, focusing on enhancing the resources selection process. This contribution proposes an Efficient Resources Selection model to determine the resources that best fit the application requirements. Hence, the model guides applications during their execution without modifying or controlling grid resources. Within the evaluation phase, the experiments were carried out in a real European grid infrastructure. Finally, the results show that not only a self-adaptive ability is provided by the model but also a reduction in the applications' execution time and an improvement in the successfully completed tasks rate are accomplished.

Keywords: Self-adaptivity, Grid Computing, Evolutionary Algorithms, Optimization, Resources Selection

Email addresses: `maria.boton@ciemat.es` (María Botón-Fernández), `mavega@unex.es` (Miguel A. Vega-Rodríguez), `francisco.prieto@ciemat.es` (Francisco Prieto Castrillo)

1. Introduction

Grid computing is an innovative distributed paradigm proposed by I. Foster and C. Kesselman [1, 2] in 90s. It was introduced as a revolutionary technique for solving massive computational problems by sharing computational power and storage capacities. The term emerges from the analogy with the electric power grids: users connect to a grid computing infrastructure and get computing power without knowing where it comes from, like the electrical power they get at home.

Several organizations and research centres are involved within a grid infrastructure by sharing their resources. These resources have different geographical locations and are grouped into Virtual Organizations (*VO*). Every *VO* refers to a set of institutions with a common goal. Furthermore, each *VO* has been associated with a particular research project. It should be highlighted that the number of resources can be increased according to the project requirements (this kind of infrastructure provides unlimited storage and computing power).

Despite the advantages of grid computing systems, there are several problems related to task management, resource discovery, resource monitoring, and resource selection. As mentioned, different centres with different administrative domains handle grid resources. This fact leads to a dynamic and changing environment: the characteristics, availability and performance of grid resources vary over time. Moreover, applications face a double heterogeneity within these infrastructures: on the one hand, grid environments are composed of heterogeneous resources (with heterogeneous hardware and software characteristics). On the other hand, there are heterogeneous resources with the same grid functionality. In addition, grid applications compete for using these non-dedicated resources. Regarding the monitoring, discovery and selection processes, the system ideally needs to know the infrastructure status in real-time for registering updated information within the *Information System* (*IS*).

For addressing these problems the *adaptation* concept is introduced as a feasible solution in grid community. This idea arises from two main issues: first, applications require real-time information about the environment for dealing with grid changing conditions. Second, the system continually requires updated information about resources (their status and availability) in order to make decisions autonomously. In recent years, several solutions based on the *adaptation* concept have been proposed. However, applying adaptation at any grid level has become a challenging topic because of the behaviour and principles of such infrastructure.

The present contribution is focused on enhancing the grid resources selection process by determining the resource set that best fit the application requirements. For that reason, it provides a self-adaptive capability to grid applications, selecting at every time the most efficient resources. The model has been designed from the user point of view without modifying the behaviour or the characteristics of grid resources. Concerning tasks, they are progressively executed (not all at once due to the fact that the model does not duplicate tasks) for profiling the resources' efficiency during the application's execution. This way, the model learns about the infrastructure's status and a suitable use of grid components is performed (considering an appropriate usage of resources without monopolizing them; grid elements can be exploited by several users at the same time). A mathematical formulation (Section 3.2) is defined for measuring resources' efficiency. This formulation is combined with an Evolutionary Algorithm (*EA*) for obtaining an efficient selection process. Specifically, the following algorithms have been combined with the proposed mathematical formulation, resulting in four versions of the model: the Variable Neighbourhood Search (*VNS*) metaheuristic [3, 4], a Cellular Automata (*CA*) methodology [5] and the Preferential Attachment (*PA*) technique [6, 7] from the Complex Network field. Moreover, in this contribution we present an enhanced version of the model based on the Scatter Search (*SS*) [8]-[10]. From now on, we denote our approach as *Efficient Resources Selection* Model (*ERS*) [11, 12] and the different versions are called respectively *ERS-SS*, *ERS-VNS*, *ERS-CA* and *ERS-PA*.

The rest of the paper is structured as follows. In Section 2 a discussion about related works is presented. Section 3 introduces the problem, by describing the model assumptions and the proposed mathematical formulation. The *ERS-SS* is described in Section 4. Section 5 summarizes the previous *ERS* versions. The evaluation of the model, including the resulting experimental data, is discussed in Section 6. This evaluation has been performed in a real European grid infrastructure. Finally, Section 7 concludes the paper.

2. Related Work

There are several researches focused on solving grid infrastructure problems by applying self-adaptation. In some cases, an adaptive system for dealing with the environmental changes is proposed (an autonomous system). In other studies, intelligent frameworks are developed for providing an efficient jobs scheduling. Finally, there are solutions that apply *adaptation* in specific grid processes (discovery, monitoring, selection, etc.).

In [13] an alternative to solve the problem of resources selection is proposed. First, the application execution starts on any resource set. Then, certain information about communication and processing times is collected periodically. With these metrics two efficiency thresholds (lower and upper thresholds) are determined. The objective is to keep the application's efficiency between these values. Resources are added or deleted based on these premises. Certain migration techniques are also applied.

Authors in work [14] propose an approach for enhancing grid infrastructures by avoiding jobs restrictions. This way, they provide an adaptive capability to applications. For that reason, the contribution describes three options to overcome those restrictions: first, an approach which implies a change on the infrastructure's design. Second, a solution in which a flexible job management is developed. Finally, a strategy focuses on fostering the cooperation between users and infrastructure.

Also the concept of *living application* emerges in grid community for solving the problems mentioned above. In this regard, the work in [15] presents a methodology for managing grid applications in an autonomous way. The methodology is based on the following principles: the application makes decisions about which tasks to do and which resources to use (these decisions are based on runtime knowledge). Thus, the application decides in an autonomous way when performing a task migration (the application requires administrative privileges).

Grid environments are increasingly used for performing long running multi-phase parallel applications. This fact has motivated the authors in [16] for developing an efficient rescheduling framework, by allowing applications to adapt to the dynamic environment. Three strategies have been designed to decide when and where to reschedule this type of grid parallel application. A similar approach is presented in [17] for long running multi-physics coupled parallel applications.

The Advanced Resource Connector (ACR) grid middleware was designed and proposed several years ago, resulting in an interesting solution for adapting this layer to new data management and storage. The version described in contribution [18] aims to solve certain problems detected in *ACR*, like system bottleneck fails. The new approach includes a layered structure (in particular 3 layers), which efficiently uses the available bandwidth as well as enables data transfer slots based on a priority system.

The research in [19] describes a migration framework in which new scheduling policies are included. In this framework, both the resources' load and the application characteristics are considered. Therefore, the novel policies are used for varying dynamically the resources' load and maintaining the infrastructure's performance. The contribution also includes an overview of self-adaptive software systems.

The work in [20] presents a framework that performs all user steps during jobs submission. Moreover, it provides mechanisms for a dynamical adaptation at runtime during applications' execution and it is based on Gridway[1]. In particular, a new application model for adapting to grid conditions is proposed. A framework architecture to support the new resulting application type is also exposed. Finally, the framework is compared with other similar approaches.

The Monitoring and Discovery System (*MDS3*) is the information services component of the OGSA-based (Open Grid Services Architecture) Globus Toolkit *GT3*[2] in which the study [21] is focused. In this research, two different self-adaptive notification algorithms to prevent the grid Index Service (*IS*) from overloading are exposed: a *sink-based self-adaptive algorithm* and an *utilisation-based self-adaptive algorithm*. The first one uses the current number of notification sinks to adjust the *IS* notification rate. The second algorithm makes decisions based on both the current number of notification sinks and the average CPU value. Furthermore, for deducing the optimal notification rate, previously compiled off-line performance benchmarks are executed in both algorithms.

AppLeS (Application Level Scheduling) [22] is a project initiated in 1996 with a specific goal: to investigate adaptive scheduling for grid computing environments. This particular project, with its corresponding results, is described along the contribution. The approach includes static and dynamic

[1]http://www.gridway.org/doku.php

[2]http://www.globus.org/toolkit/

resources information, performance predictions and scheduling techniques for allowing applications adaptation during their execution.

Finally, several existing self-adaptive mechanisms for grid computing are collected as a survey in [23]. Specifically, seven grid systems applying adaptation are presented, analysed and compared. In the end, certain considerations and recommendations for reaching both an autonomous operation and an autonomous management in grid systems are suggested.

All these works, although are focused on solving different grid problems considering *adaptation*, have a common feature: they are designed from the system's viewpoint. That is to say, the described solutions modify the behaviour of certain grid elements - by applying scheduling or migration techniques, by proposing new notification algorithms, by developing intelligent frameworks, etc. However, our proposed *ERS* model is designed from the user point of view, in other words, taking into account users limitations and operations. Furthermore, it does not apply scheduling techniques, it does not control grid components and it does not change their behaviour. It just guides grid applications during their execution, by determining the most efficient resources that fit the application requirements.

The following Section includes a detailed description of the *ERS* model characteristics, assumptions and formulation.

3. Problem Statement

As said before, a grid computing environment provides unlimited computational and storage power. Namely, the main goal of grid systems is to use remote resources that enable users to perform very large tasks that could not be addressed in their machines or work centres. For that reason, one of the fundamental interests of grid communities lies in the efficient use of grid resources. However, the dynamic nature of these distributed systems implies that these resources may appear and disappear unpredictably. Thus, scalable and efficient mechanisms are necessary for an automated discovery and efficient selection of resources.

In this sense, we present an Efficient Resources Selection model which allows applications to self-adapt to grid changing conditions. Moreover, our model is focussed on improving the infrastructure throughput. Consequently, both premises (*self-adaptation and throughput*) are expected to induce, first, an execution time reduction for grid applications and, second, an increment of successfully completed tasks rate, due to the fact that inefficient resources are avoided.

3.1. The Model Assumptions

The *ERS* model is designed from the user point of view, which is one of its main characteristics. For that reason, the model applies users operations for obtaining information about resources in real-time. By using this gathered information, the resources efficiency is continually measured during applications execution. Next, some grid specifications are briefly exposed for a better understanding (Table 1 summarizes the basic grid elements).

Table 1: Basic elements in a typical grid infrastructure.

Acronym	Grid Name	Function
UI	User Interface	User's access point to a grid infrastructure.
CE	Computing Element	Scheduler located in a grid site which handles the jobs' queue.
WN	Worker Node	Component of a grid site with computational capacity. Jobs are executed in these elements.
WMS	Workload Management System	Main Scheduler which deals with load balancing and selects CEs.
RB	Resource Broker	Physical machine in which the *WMS* is installed.
BDII	Berkeley Database Information Index	The global information system of a grid environment.
LFC	LHC File Catalog	Distributed file catalog including an unique global name space.

Notice that in a typical grid infrastructure (as shown in Figure 1) resources are grouped in two different sites: Resources Operation Centre (*ROC*) and Resources Centre (*RC*)[3]. The *ROC* maintains important information systems as the *BDII* (*Berkeley Database Information Index*) and the *LFC* (*LHC File Catalog*). The interaction between users and a grid environment occurs

[3]A grid site, which was referenced in Table 1, is considered as a Resource Centre (*RC*).

through the access point known as *User Interface* (*UI*). By using the *UI* command line, jobs[4] are launched into execution; previously a job description file is built for every application task. Then, jobs are handled in the corresponding *ROC* by the *Workload Management System* (*WMS*), whose purpose is to accept the submitted jobs, to allocate them in the most suitable *Computing Element* (*CE*), to record their status and, finally, to return the generated output. The machine where this service is installed is known in grid terminology as *Resource Broker* (*RB*). Regarding the *CEs*, they are a type of scheduler at the *RC* level that decides in which *Worker Node* (*WN*) tasks will be executed. For selecting a *CE* a process denoted as *match-making* is applied, according to which the availability and goodness of resources are taken into account along with tasks requirements.

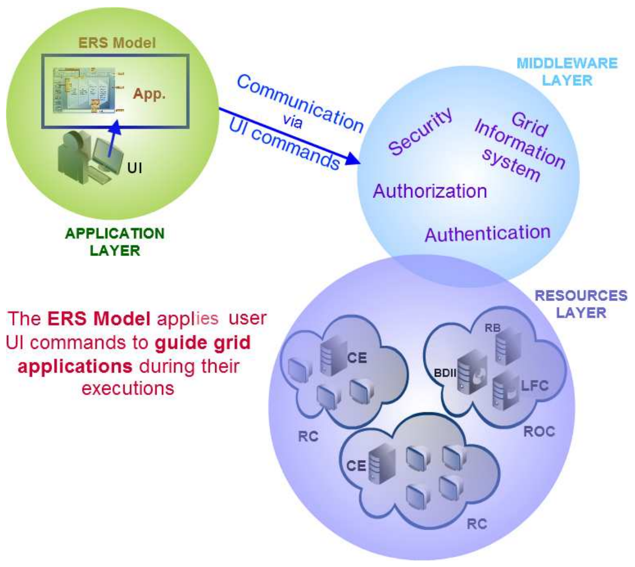


Figure 1: Representation of a typical grid architecture. The proposed *ERS* model is located in the *UI*. Only user commands are applied for monitoring the resources' efficiency.

Furthermore, grid philosophy enables users to specify which *CE* will manage their tasks. Considering this circumstance along with previous assump-

[4]A job is a grid task defined by using a specific description language known as *JDL* (Job Description Language) [24]. With this language, users describe the task or tasks (belonging to a particular application) to be performed in the grid infrastructure. Hence, a job wraps users' application tasks, allowing them to run in a *WN*.

tions, we have decided to measure and to classify *CEs* based on their efficiency value (see the model formulation in the following section).

3.2. The Model Formulation

This Section describes how the model measures the resources' efficiency. The model manages two workspaces during the application execution: a task space $\mathbf{J}$, consisting of n independent and parallel tasks to be processed (tasks only differ in their input parameters value), and a heterogeneous resource space $\mathbf{R}$, which contains the m available *CEs* from the corresponding infrastructure. Notice that the model has been defined for supervising the execution of parametric sweep applications, in which all tasks have similar characteristics and behaviour (this fact reduces the problem complexity).

Then, the model guides applications selecting an efficient resource $rt_\alpha \in R$ for every task $t_\alpha \in J$ (the model finds the best *CE* (rt_α) to execute the next task (t_α); we do not try to find the best pair (rt_α, t_α)). For speeding up the model learning a subset of J is launched at the beginning of the application execution. This subset is denoted as P_α and has associated a subset of R denoted as RP_α. These resources are selected in a random way because at that moment there are not efficiency metrics (i.e. all of them are initially considered as efficient).

Once the P_α set is launched into execution, tasks are monitored independently. Every task t_α has assigned a *lifetime* lt_j to be executed. The corresponding resource rt_α should complete the task within that time. Hence, the efficiency value of a resource is updated when the corresponding t_α is finished or when its lt_j ends.

Concerning the efficiency metrics, several parameters are defined within the model. First, the *processing time* $T_{i,j}$ of a particular *CE* is illustrated in Eq.(1). This parameter depends on the *CE communication time* $Tcomm_i$ with other grid services[5] and on the *computation time* $Tcomp_{j,i}$ of task j, which is allocated in resource i.

$$T_{i,j} = Tcomm_i + Tcomp_{j,i} \ . \tag{1}$$

Next, the time consumed θ_i by resource i with respect to task lifetime lt_j is calculated as shown in Eq.(2). Furthermore, the resource's *processing time*

[5]As the model guides parametric sweep application (all tasks are identical) we consider that $Tcomm_i$ mainly depends on the corresponding *CE*.

$T_{i,j}$ is applied. A value closer to 1 implies that the corresponding task was performed in a faster way.

$$\theta_i = (lt_j - T_{i,j})/lt_j \ . \tag{2}$$

Finally, the fitness F_i of a resource is based on its rate of successfully completed tasks ϵ_i and on its θ_i value. Furthermore, both parameters have associated a *weight* or *relevance value* denoted respectively as a and b (see Eq.(3)). These *relevance values* are specified by users to determine the requirements of their experiments. Every resource with a fitness value close to 1 is considered as an efficient one.

$$F_i = (a \cdot \epsilon_i + b \cdot \theta_i)/(a + b) \ . \tag{3}$$

4. Developing the ERS Model Applying an Evolutionary Algorithm

This Section introduces the combination between the mathematical formulation and the *Scatter Search* method, resulting in the *ERS-SS* model version.

Figure 2 represents the functional diagram which comprises the behaviour of our proposed model. It is possible to distinguish three phases within that execution diagram: processes with dashed line (Phase 1), processes with dotted line (Phase 2) and processes with solid line (Phase 3).

The processes with dashed line compose the first phase, denoted as *Preparing the environment*, in which the main objective is to adjust the environmental conditions of the model. For that reason, spaces J and R are composed in this stage. Notice that users provide certain generic information about their scientific applications such as input data, output data, execution time of a particular task, tasks' requirements, etc (all this information is gathered through command line and templates). Furthermore, the model queries the *Information System IS* (by applying user commands) for determining the components of space R (i.e. for discovering the available resources). Concerning the parameters handled by the model, both the real workload value of available resources and the lifetime lt_j of every application task are also initialized. The resources workload is calculated by querying the *IS* as it was done before (this workload value will be continually updated in Phase 3). The lifetime lt_j will be initialized considering the application information. Finally, the model prepares the P_α subset.

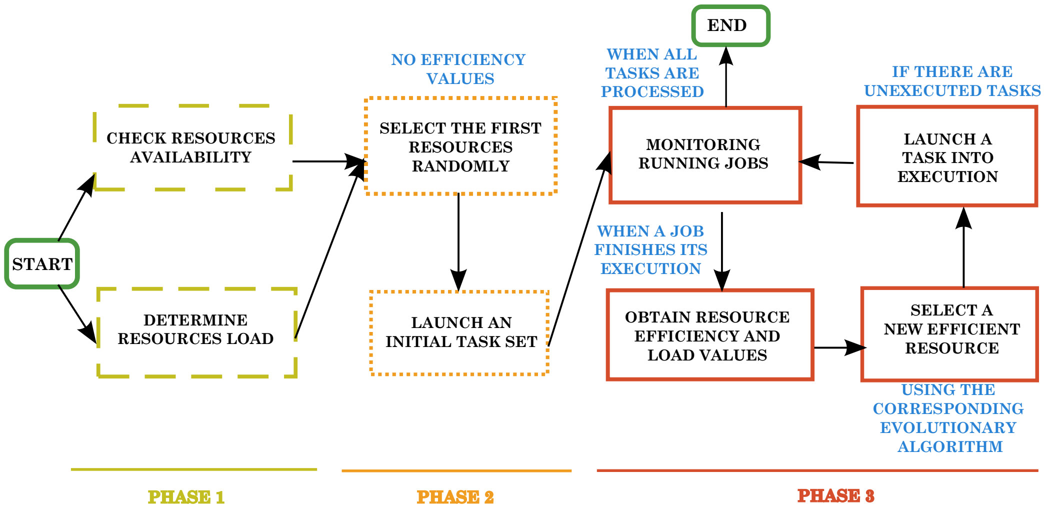

START
CHECK RESOURCES AVAILABILITY
DETERMINE RESOURCES LOAD
NO EFFICIENCY VALUES
SELECT THE FIRST RESOURCES RANDOMLY
LAUNCH AN INITIAL TASK SET
END
WHEN ALL TASKS ARE PROCESSED
MONITORING RUNNING JOBS
WHEN A JOB FINISHES ITS EXECUTION
OBTAIN RESOURCE EFFICIENCY AND LOAD VALUES
IF THERE ARE UNEXECUTED TASKS
LAUNCH A TASK INTO EXECUTION
SELECT A NEW EFFICIENT RESOURCE
USING THE CORRESPONDING EVOLUTIONARY ALGORITHM
PHASE 1
PHASE 2
PHASE 3

Figure 2: Scheme of the *ERS* model in which the self-adaptive capability is represented. The model guides the application avoiding inefficient, inoperative and overloaded resources.

Next, the application execution is started (processes with dotted line) within the second phase (named *Initialization*). Once spaces J and R are prepared and the corresponding P_α set is ready for execution, the model selects a set of resources RP_α for performing P_α. As stated, this subset $RP_\alpha \subset R$ is chosen in a random way because there are not efficiency metrics yet. That means, as it is unknown how efficiently resources perform, the model considers that all of them execute tasks in the same way (i.e. initially it is considered that all of them have the same efficiency value). It should be highlighted that P_α and RP_α follow a many-to-one relationship (several tasks could be managed by the same *CE*), so that, it is possible to find repeated resources in the RP_α set. After that, P_α is launched into execution.

Finally, the processes with solid line included in the third phase (the *Adaptive Execution* phase) are used to provide a self-adaptive capability to the corresponding application. Once P_α is launched, the tasks that compose this set are monitored (as in Phase 1, the *IS* will be queried to determine the grid status of each task). When a task t_α finishes[6] its execution, the

[6]There are three possibilities: when the task's lifetime ends, when it is cancelled by the infrastructure or when it ends its execution successfully.

resource's information (workload, $T_{i,j}$, etc.) is updated and its efficiency value F_i is obtained. Moreover, the real workload value of every available *CE* is updated (the model takes into account the current infrastructure's status in its decisions). After that, by applying an Evolutionary Algorithm (*EA*), a resource is efficiently selected for a new task; this task is launched into execution (its lifetime has been previously initialized). The processes with solid line are repeated until all tasks $t_\alpha \in J$ are performed.

The following subsection describes how the Scatter Search method is used in the selection process, resulting in a self-adaptive process.

4.1. The ERS based on the Scatter Search

The Scatter Search [8]-[10] is an evolutionary algorithm used for solving a wide range of hard optimization problems. Mainly, it is based on combining solutions for creating new enhanced ones. This combination applies systematic selections over a *Reference Set* (small set of solutions). This set includes the *good solutions* that have been found during the search process. The *goodness* of a particular solution is based on quality and on diversity criteria.

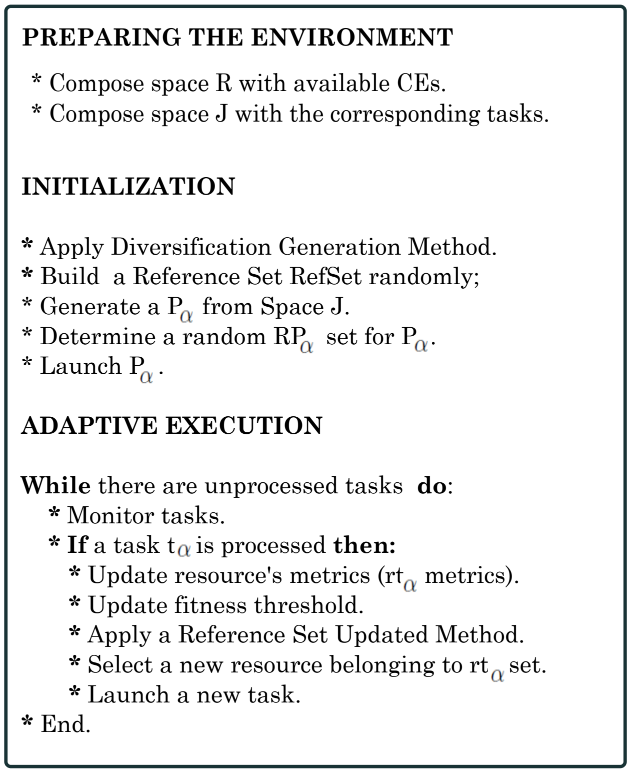
As far as the *ERS* model concerned, several components have been included for adapting the Scatter Search method to the assumptions established in previous sections. It should be highlighted that in this *ERS* version we are focused on applying the ranking part of the *SS* algorithm during the selection strategy. The next three elements are also part of the efficient selection process. Notice that combining the two following sets (S_Q and S_D) the *Reference Set* is constituted.

- **A Quality Set** S_Q consisting of the q most efficient resources.

- **A Diversity Set** S_D including the h most diverse resources with respect to S_Q. The diversity value is based on efficiency and geographical criteria. First, resources from R_D ($R_D = R - S_Q$) are classified by site membership (geographical location). Then, every diversity value is calculated considering this geographical classification along with the site membership of quality resources. Next, all resources from R_D are sorted from higher to lower diversity value. The first h resources that exceed the fitness threshold will compose the S_D set. If we have less than h resources that exceed the threshold, we select the remaining ones in a random way from R_D set, without duplicating resources.

- **A Fitness Threshold** that is applied during the composition of S_D. It is the average fitness value of all evaluated resources. Every time a resource rt_α processes a task and its efficiency value is updated, the model updates the threshold.

Concerning the *ERS-SS* execution pseudocode, the main steps within each phase (*Preparing the environment, Initialization and Adaptive Execution* in Figure 2) are described in Figure 3. Once the model's environment is prepared, two methods owned to *SS* are applied for obtaining a *Reference Set*. Owing to the fact that there are not efficiency metrics, S_Q and S_D are formed in a uniform random way. Next, the P_α set is composed. Tasks belonging to that set are also selected randomly from space J. Similarly the RP_α set is generated at that moment in a random way. The model initializes tasks lifetime and launches P_α into execution.



```
PREPARING THE ENVIRONMENT

 * Compose space R with available CEs.
 * Compose space J with the corresponding tasks.

INITIALIZATION

* Apply Diversification Generation Method.
* Build  a Reference Set RefSet randomly;
* Generate a $P_\alpha$ from Space J.
* Determine a random $RP_\alpha$ set for $P_\alpha$.
* Launch $P_\alpha$.

ADAPTIVE EXECUTION

While there are unprocessed tasks  do:
   * Monitor tasks.
   * If a task $t_\alpha$ is processed then:
     * Update resource's metrics ($rt_\alpha$ metrics).
     * Update fitness threshold.
     * Apply a Reference Set Updated Method.
     * Select a new resource belonging to $rt_\alpha$ set.
     * Launch a new task.
* End.
```

Figure 3: The main instructions of the *ERS-SS* execution flow are specified in this figure. It is possible to distinguish the different phases that compose the model.

Within the *Adaptive Execution* phase tasks are monitored independently. When a task t_α is processed (it finishes its execution) the efficiency metrics of the corresponding resource rt_α are updated. Next, the model applies a *Reference Set Updated Method*. Within this method, S_Q and S_D are composed as mentioned above. The elements of both sets will be part of the new updated *Reference Set*. A new efficient resource is selected from *Reference Set* in case space J is still unprocessed. Overloaded resources as well as unavailable resources are avoided using this adaptive execution. This efficient resource should belong to the previous resource set: if the previous resource rt_α belongs to the S_Q set, then, the new one belongs to the same set. On the contrary, if it belongs to S_D the new resource is selected from that resource set. Once a new resource is chosen, the model sends a new task into execution. This adaptive phase is repeated until the whole space J is processed.

5. Previous ERS Model Versions

5.1. The Preferential Attachment Model Version

The *Preferential Attachment* (*PA*) [6, 7] technique is a famous algorithm within the Complex Networks field. Barabási and Albert introduced it to explain the power-law degree distribution in complex networks. The main idea is that nodes in a Complex Network (a graph with non-trivial topological features in which nodes represent the dynamical units) with a higher degree acquire new nodes more easily than the remaining nodes. These nodes with a higher degree are known as *hubs*. In a group of friends, *hubs* are the most popular ones.

A paradigmatic example of a preferential attachment process is the Internet growth, from a webpage point of view. In this regard, a new webpage will be linked to a highly visited webpage with a higher probability than to another page with few visits. Then, it is possible to conclude that a new node i is linked to an existing one j with a probability proportional to the node degree (it is the total number of links of a node).

In the *ERS-PA* [11] version the following assumptions are considered for applying this complex algorithm in the selection process.

- The resources belonging to space R will be represented as nodes in the corresponding complex network.
- This complex network is built at runtime.

- The degree of a particular node represents the number of times this resource has been selected during application execution.
- The probability of connecting a new node with an existing one depends on resource's efficiency.
- The efficiency (E_i) of a resource is based on both the resource relative degree (k_i/k_{max}) and the resource's fitness value (F_i; Eq.(3)):

$$E_i = (k_i \cdot F_i)/k_{max} \ . \qquad (4)$$

5.2. The Variable Neighbourhood Search Model Version

The Variable Neighbourhood Search (*VNS*) [3, 4] is a metaheuristic used for solving optimization problems by applying a search process in a specific space of solutions. The main idea is to change the neighbourhood structure during the local search. The steps of *VNS* are represented in Figure 4.

```
INITIALIZATION

* Select a set of neighbourhood structures $N_k$;
* Specify an initial solution x;
* Choose a stopping condition;

ITERATIONS

Repeat until the stopping condition is met:
    * k ⟵ 1;
    * Repeat until k = $k_{max}$:
        * Shaking: generate a random solution x' from the kth neighbourhood of x;
        * Local Search: apply a local search method considering x' as the initial
          solution; Denote with x" the resulting local optimum;
        * Move or not:
            * if x" is better than x do
                * x = x";
                * k = 1;
            * else: k = k + 1;
* End.
```

Figure 4: Main rules that govern the *VNS* algorithm.

According to the information described in Figure 4 the *VNS* looks for a global maximum or minimum solution, depending on the problem. In our case, as we want to find the most efficient resources, the model tries to maximize the fitness of grid resources (get fitness values close to 1). Furthermore, considering the fact that we sent P_α into execution, in *ERS-VNS* the efficiency of the whole RP_α set is measured (see Eq.(5)).

$$F(RP_\alpha) = (\sum_{i=1}^{d} F_i)/d \ . \tag{5}$$

The fitness is calculated as specified in Eq.(3) for every resource rt_α belonging to RP_α. The cardinality of this resource set is denoted as d. Hence, an initial solution x in the proposed *ERS-VNS* version is the RP_α set. Then, the solution x' is obtained after performing a mutation or shaking process over x. Finally, x'' is a new optional solution (a new RP_α set).

In this version, we introduce two parameters that are handled by every neighbourhood structure: p_k and q_k. In a particular neighbourhood structure k^{th}, parameter p_k quantifies the variation/difference between the initial solution x and the mutated solution x'. This parameter varies in an increasing way from a structure to another. Moreover, parameter q_k indicates the search range applied during the local search. The value of such parameter depends on the number of available resources. These resources are ordered from higher to lower efficiency value.

In order to enhance the behaviour of *ERS-VNS* a fitness threshold is also included. This threshold is used during the mutation/shaking process for determining which resources should mutate (those which are considered as inefficient resources).

5.3. The Cellular Automata Model Version

A Cellular Automata (*CA*) [25] is a discrete model composed by a grid of cells, each of them with a finite set of status. Moreover, each cell has associated a neighbourhood that is constituted by the closest cells in the grid.

In this version of the model, *ERS-CA* [12], the grid infrastructure is modelled as a *CA*, in particular as a Cellular Automata Network [5]. The *CEs* are represented as cells of a *CA* and they are grouped in subnetworks by considering geographical criteria. The way to locate resources in the *CA* is by establishing that physical neighbours in the infrastructure would be neighbours in the resulting *CA*. The number of subnetworks depends on the amount of available resources; a minimum cardinality value of 3 resources is fixed in all the subnetworks.

Each cell in *ERS-CA* has three possible states: *living, dead* and *inoperative*. The model considers that a *living* cell represents an efficient resource. In contrast, when a resource is classified as inefficient the corresponding cell has a *dead* status. Finally, the unavailable resources are considered as *inoperative* cells. In this version, the efficiency of a resource is calculated by using Eq.(3).

The resources workload is used within the governing rules for detecting when a resource is overloaded. In every subnetwork, and during the selection process, a workload threshold is applied to determine the surviving cells. In this regard, only the two least loaded cells will survive in each subnetwork. Next, the governing rules of *ERS-CA* are described:

- The two cells with minimum workload value are considered *living* cells.
- When a cell exceeds the workload threshold dies.
- A cell with all its neighbours dead survives.

Finally, with all the *living* cells the model generates a candidate set. The following rules are applied:

- Selective Pressure: the most efficient 50% of candidate cells promotes as a solution.
- Scout Resource: an unexplored resource is selected in a random way by the model to be part of the solution.

6. Experimental Evaluation

The Efficient Resources Selection model has been tested in the European Grid Infrastructure (*EGI*[7]). This platform coordinates different grid projects like the *National Grid Initiative* (*NGI*). Specifically, we perform the real experiments in the Spanish National Grid Initiative (*ES-NGI*)[8]. We were affiliated to the Ibergrid[9] *VO*, an organization with 30 CEs approximately.

[7]http://www.egi.eu/
[8]http://www.es-ngi.es/
[9]http://www.ibergrid.eu/

As stated, during the model design two objectives were fixed: a reduction in the application's execution time and an increase in the successfully finished tasks rate. We consider that both goals are interesting topics for grid users. The *ERS* model monitors the resources' efficiency during the whole applications execution, avoiding those resources that are overloaded, inoperative or inefficient. We expect that the combination of discarding inefficient resources and selecting the most efficient ones leads to a reduction in the application execution time. Furthermore, with this fact (exploiting the throughput) we aim to improve the successfully finished task rate.

The way to determine if the objectives have been accomplished is by defining two scenarios in which we also analyse the model behaviour. In this regard, the first scenario is used for studying the model learning and how it is influenced by different parameters. The second scenario is designed to determine if the model performs properly for applications with a large number of tasks. In both scenarios, we compare our approach with the standard selection technique in European grid infrastructures, which are based on gLite[10] middleware. This technique includes a mechanism known as *match-making* [24] based on which the *WMS* distributes jobs on the grid infrastructure. This mechanism first chooses those available *CE* that fulfil the user requirements and that are close to the specified input files. Then, it selects a *CE* with the highest rank value. Notice that the default ranking mechanism of *WMS* takes into account the resource's response time. The *match-making* is a symmetric algorithm in which the *CE* evaluation goes in both directions due to the fact that both the requirements specified by users and the requirements defined by the *WMS* are considered. Finally, we refer to this standard technique as *Traditional Resources Selection* (*TRS*).

Also in this Section certain circumstances of the model and some application's characteristics have been analysed. They are discussed in Sections 6.3, 6.4 and 6.5. In all these studies the new *ERS-SS* versions have been assayed considering the characteristics fixed in scenarios 1 and 2. Different applications have been considered in our experiments: Runge-Kutta (sections 6.1, 6.2, 6.3 and 6.4), Gaussian, LU Decomposition, and FFT (section 6.5, analysis of workflows). Finally, parameters a and b are fixed at 60% and 40% respectively; we give a slight relevance to the number of successfully finished tasks because we think this is an important issue for grid users. Notice that

[10]http://glite.cern.ch/

10 real executions of each version have been performed in every test. Hence, the results shown in the corresponding figure for each version (the *TRS* and the *ERS* versions) are the average values of this set of executions.

6.1. Scenario 1: Analysing the Model Learning

In this scenario we want to determine how the size of P_α influences the model learning. Thus, the size of space J is fixed at 200 tasks while in every test the size of P_α varies from 5 to 40 tasks (as shown in Table 2).

Table 2: Configuration characteristics in scenario 1.

	P_α	Space J	Algorithm
Test 1	5	200	*PA, VNS, CA, SS and TRS*
Test 2	10	200	*PA, VNS, CA, SS and TRS*
Test 3	13	200	*PA, VNS, CA, SS and TRS*
Test 4	20	200	*PA, VNS, CA, SS and TRS*
Test 5	40	200	*PA, VNS, CA, SS and TRS*

From the obtained results (Figure 5) it is possible to conclude that all *ERS* versions get better execution times with respect to the standard European grid technique *TRS*. In particular, the proposed approach described in this contribution, *ERS-SS*, reaches the best execution times. In this case, when the size of P_α is increased the model achieves a higher execution time difference with respect to *TRS*. This is due to the fact that the number of resources evaluated at the beginning of the application execution grows in each test (this situation motivates a fast learning).

However, the *ERS-PA* version gets its minimum values in the first tests (5, 10 and 13). In the last tests (20 and 40) probably the *hubs* in the complex network become overloaded more quickly because of the quantity of tasks sent at the same time. When this fact occurs, the model takes a long time to appreciate this overload because the efficiency (as indicated in Eq.(4)) depends on both the fitness and degree values (the degree of a *hub* is very high so it counteracts a low/bad fitness).

Concerning *ERS-VNS*, it gets a constant execution time difference with respect to *TRS*. In the last three tests (13, 20 and 40) the *ERS-VNS* reaches higher execution time values than in the first two tests. This is due to the fact that for large sizes of P_α there are less changes of neighbourhood structures, so the model is far from obtaining the global optimum resource set.

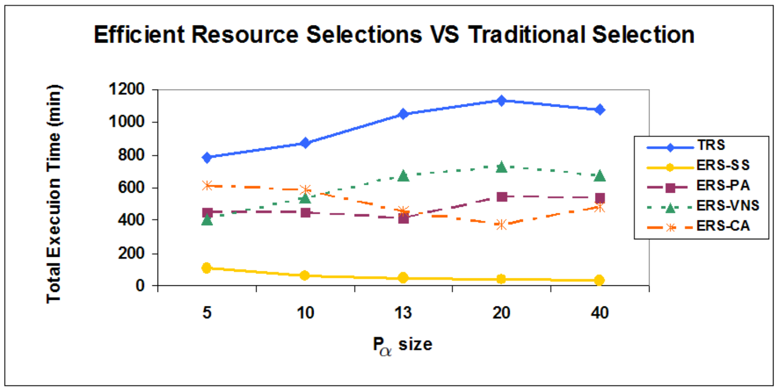


Figure 5: The results obtained by the different versions after performing the first scenario. The *ERS-SS* improves the execution time with respect to the previous *ERS* versions and *TRS*.

Regarding the *ERS-CA*, for high sizes of P_α there is a fast model learning because more resources are selected/evaluated from the first moment; this way the model avoids resources overload and achieves a complete efficiency rating in a shorter time. Finally, the successfully finished tasks rate is also improved in all *ERS* versions as shown in Figure 6. Notice that in every test (i.e., in every size of P_α), results are represented following the order *TRS*, *ERS-SS*, *ERS-PA*, *ERS-VNS* and *ERS-CA*. The *ERS-SS* obtains the best tasks rate in average.

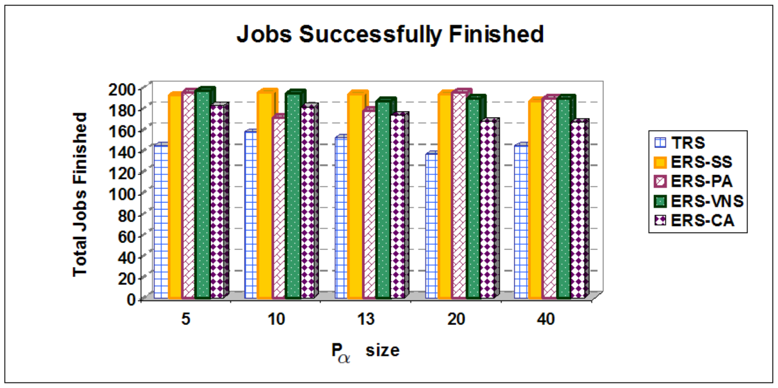


Figure 6: Representation of the successfully finished tasks rate for the different versions in all tests.

6.2. Scenario 2: Analysing Large Sizes of J

This second scenario has been defined to determine the range of applications in which it is appropriated to use the *ERS* model. For that reason, the size of space J is varied in every test from 50 to 500 tasks (see Table 3). The size of P_α is fixed at 10 tasks.

Table 3: Configuration characteristics specified for the second scenario.

	P_α	Space J	Algorithm
Test 1	10	50	*PA, VNS, CA, SS and TRS*
Test 2	10	100	*PA, VNS, CA, SS and TRS*
Test 3	10	200	*PA, VNS, CA, SS and TRS*
Test 4	10	300	*PA, VNS, CA, SS and TRS*
Test 5	10	400	*PA, VNS, CA, SS and TRS*
Test 6	10	500	*PA, VNS, CA, SS and TRS*

In Figure 7 the results that have been obtained in the 6 different tests are represented. *TRS* gets the highest execution time in all tests. The *ERS-PA*, *ERS-VNS* and *ERS-CA* versions obtain execution times close to *TRS* in the first three tests (50, 100 and 200). *ERS-VNS* and *ERS-CA* have a similar behaviour in the whole scenario, improving the execution time difference with respect to *TRS* as the size of space J is increased.

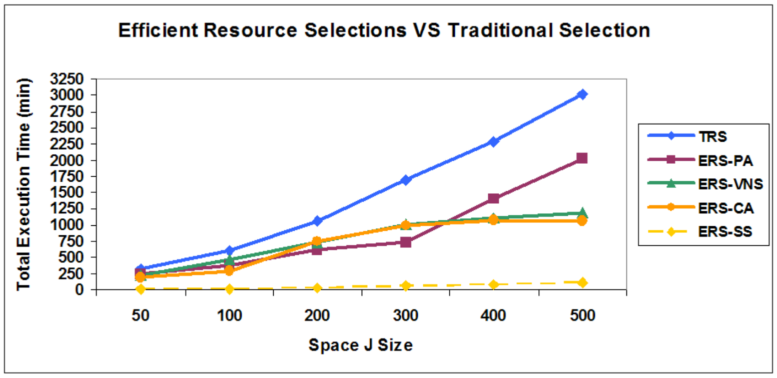


Figure 7: Comparative graphic from Scenario 2 in which we represent the results obtained for every version. *ERS-SS* gets the best execution time in all tests.

Concerning the *ERS-PA* version, it does not improve its execution time from a size of 300 tasks to the last one (500), although these time values are still better than the execution time of *TRS*. Finally, the *ERS-SS* version achieves a great execution time difference with respect to *TRS* and also presents a significant time difference with respect to the other *ERS* versions. Finally, as shown in Figure 8, the successfully finished tasks rate is also improved in all the *ERS* versions with respect to *TRS*.

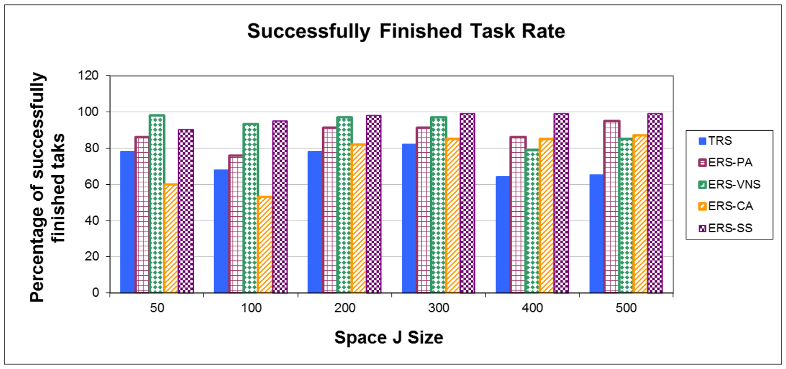


Figure 8: Percentage of successfully finished tasks for the five versions in Scenario 2. It is possible to observe that *ERS-SS* has the most constant rate in all tests.

From both scenarios we can conclude that the two objectives fixed (reducing the application's runtime and improving the rate of successfully finished tasks) have been accomplished, so that, the *ERS* model is a feasible solution for grid applications.

6.3. Scenario 3: Analysing Tasks Replication

In the two previous scenarios, the size of space J has been fixed at a particular value x, although some tasks have been processed without success (which is an usual situation in grid infrastructures). However, most of users need to finish all tasks in a successful way. For that reason, in this scenario we modify the *ERS-SS* model for considering tasks replication, ensuring a successful completion of space J.

Thus, the *ERS-SS* model monitors tasks as specified in Section 4 but now, when a task t_α is processed the next rules are applied:

- If the grid status of t_α is *Done*, the model selects an efficient resource and launch a new task.
- If the grid status is *Aborted* or the *lifetime* of t_α ends before it is successfully completed, the model selects two efficient resources: one for replicating t_α and another for executing a new task.

From now on, we denote this version as *ERS-SS-RE* (*ERS-SS* with replication). Next, for evaluating this new behaviour of the *ERS* model we perform the previous scenarios (1 and 2) with the replication approach. We also maintain the parameters configuration specified in Tables 2 and 3. First, the data resulting from performing scenario 1 are discussed. In this case, Figure 9 shows that there is a slight execution time difference between *ERS-SS* and *ERS-SS-RE* (i.e. an average execution time difference of 25 minutes between both *ERS-SS* versions. The replication approach takes 25 minutes more than the non-replication one). Furthermore, the execution time is much better than in the *TRS* version.

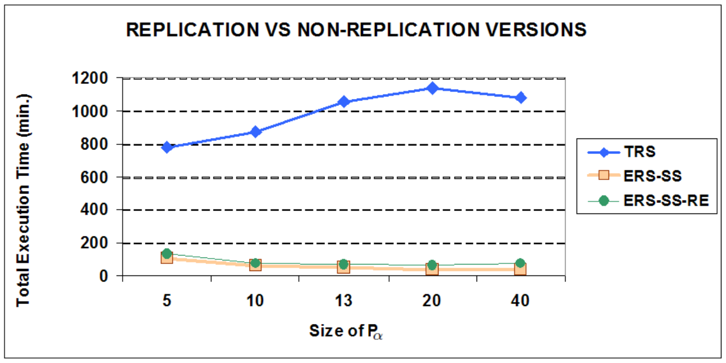


Figure 9: Results that have been obtained during the first scenario when applying replication in the *ERS-SS* model.

Now, in Figure 10 we compare our replication approach with a *TRS* version in which the replication option is also included (denoted as *TRS-RE*). Notice that the standard grid selection with replication is even slower than *TRS*.

Concerning the second scenario (see Figure 11), notice that the *ERS-SS-RE* version gets again close values to *ERS-SS*. The average execution time difference between the replication and non-replication *ERS* versions is about 7 minutes. With a higher size of space J the model classifies better the grid resources, implying a low replication rate. However, as shown in Figure 10 and Figure 12, the *TRS-RE* spends more execution time for processing the whole space J. Then, we can conclude that it is possible to apply replication in our *ERS* model without affecting the objectives fixed.

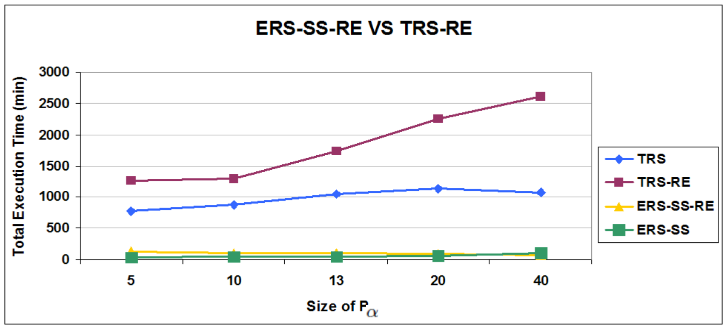


Figure 10: In this case it is included the *TRS* considering replication. It is possible to observe that *TRS-RE* spends even more execution time than *TRS*.

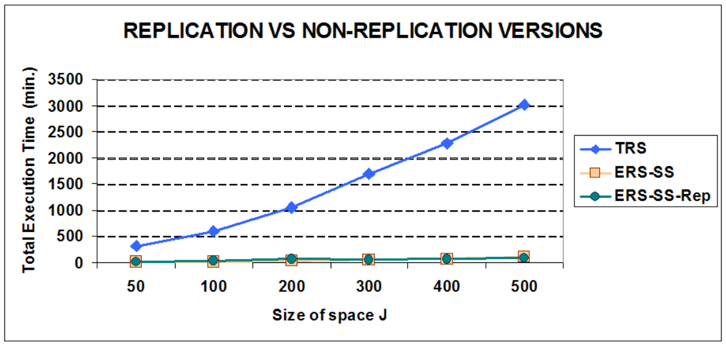


Figure 11: Results obtained during the second replication scenario when applying replication in the *ERS-SS* model for large productions. There is a slight execution time difference between the two *ERS-SS* versions.

6.4. Scenario 4: Analysing Free Slots

In this fourth scenario we intend to simulate load balancing by considering *free slots* during the model execution. In that sense, as the model is defined from a user point of view we do not implement any load balancing method. In our case, the *ERS* model measures the actual resources' workload and uses this value to dismiss resources within the selection process.

During the initialization phase, the model calculates the workload for every available *CE*. The required information for measuring this value is obtained by querying the *Information System* (*IS*). This initial workload value is used during the random selection of RP_α (for executing the initial task set P_α). This way, the overloaded resources are avoided from the beginning of application's execution.

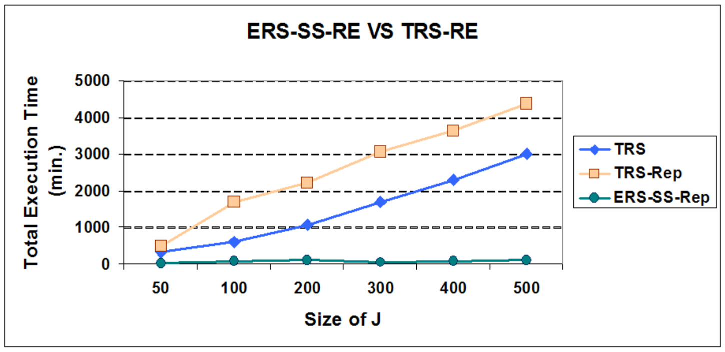


Figure 12: Results obtained in the second scenario with the replication option in the standard grid selection *TRS*.

Next, every time a resource rt_α processes a task, the workload of each resource $rt_\alpha \in R$ is updated. This value is considered within the *Reference Set Updated Method* for obtaining the quality and diversity sets (S_Q and S_D). An **efficient** resource will be part of S_Q if it is **not** considered as overloaded. In the same way, a **diversity** resource will be part of S_D if it is **not** overloaded. The model considers that every resource with a workload value close to 1 (100%) is overloaded.

We perform the corresponding tests of scenarios 1 and 2 to analyse the behaviour of such *ERS* approach (*ERS-SS-Slots*). In this case, we compare this version with the original *ERS-SS* version and the standard European grid technique *TRS* (applying *free slots* in *TRS* would be a way of modifying the standard grid selection technique used in European grid infrastructures, for that reason we do not consider it). The resulting data are shown in Figure 13, in which it is possible to observe that *ERS-SS* with *free slots* criterion takes more time to execute the application than *ERS-SS*. The average execution time difference between the two *ERS* versions is about an hour.

Next, Figure 14 includes the three different versions when using the parameters configuration fixed in scenario 2 (Table 3 in Section 6.2). The two *ERS* versions improve the execution time achieved by *TRS*. However, the *free slots* version spent more execution time, as in the previous graphic. The average execution time difference is about 2 hours between the *ERS* approaches.

From the results that have been obtained in the whole scenario, it is possible to determine that an efficient resource handles its tasks queue faster

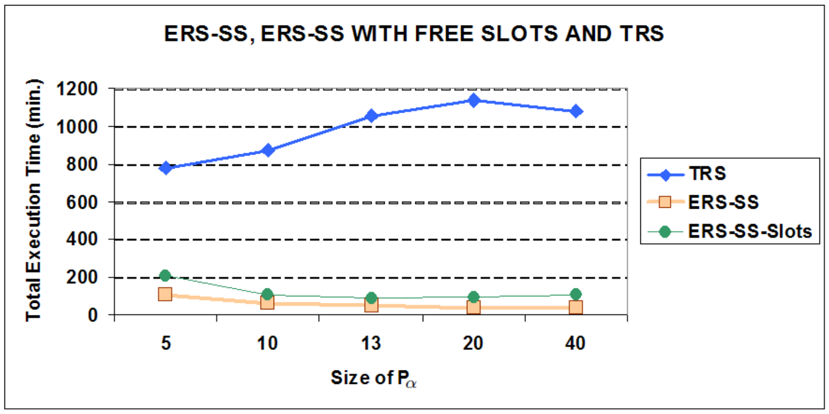


Figure 13: Comparative graphic for scenario 4 using the parameter configuration established in scenario 1.

than another with a lower efficiency value, even when the more efficient resource is overloaded and the other one has a lesser workload value. As a conclusion, it is beneficial to exploit the best resources when they are available.

6.5. Scenario 5: Analysing Data Dependence Workflows

Grid applications can be classified in two main types of workflow structures [26]: *DAG* (Directed Acyclic Graph) and *non-DAG* (Non Directed Acyclic Graph). In *DAG-based* workflow, the workflow structure can be categorized into *sequence*, *parallelism*, and *choice*. *Sequence* represents an ordered series of tasks, in which a task starts its execution when a previous one has been completed. *Parallelism* is composed by tasks which are performed concurrently. In *choice* a task is selected to be executed when its associated conditions are true. *non-DAG* workflows include not only all the patterns defined previously but also the *iteration* structure. In this pattern some sections of workflow tasks are allowed to be repeated.

Initially, the proposed model was defined for guiding parametric sweep applications (*non-DAG parallelism* workflow) constituted by independent and parallel tasks; these tasks only differ in the input parameters value. However, we consider an important issue allowing the model to guide applications with a *DAG-based* workflow. In Figure 15 we represent the chains of processes that form the workflow used in this scenario.

Three well-known applications are used in the workflow: *Gaussian*, *LU*

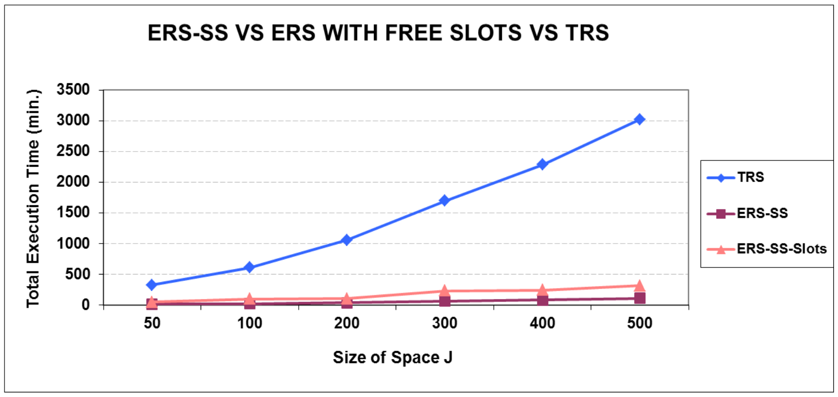


Figure 14: Results obtained when performing scenario 4 with the *free slots* version and considering the parameters' configuration of scenario 2.

(Lower/Upper) Decomposition and *FFT* (Fast Fourier Transform) algorithms. The flow consists of nodes connected by directed arcs. Each node (except *Launch* and *Report*) represents a single computational task; this task has a set of input and output arcs. Nodes connected with the *Launch* node receives control directives to initiate the computation. The remaining nodes receive input data from other nodes through its input arc(s). Finally, nodes connected with the *Report* node send their computed solution along the output arc(s). Notice that some *FFT* nodes are used to initialize the successor *Gaussian* nodes. The number of *Gaussian* nodes launched into execution at the beginning of execution is equal to P_α size.

The *ERS-SS* model measures the efficiency of grid resources in a similar way for *non-DAG* and *DAG-based* workflows. However, in *DAG-based* workflows every time a new task is sent into execution the following rules should be considered to determine its type (*Gaussian*, *LU* and *FFT*).

- Every *Gaussian* task transfers its solution to a *LU* task.
- A *LU* task sends its output data to a *FFT* task.
- *FFT* tasks initialize the computation of *Gaussian* tasks as necessary.

Once the new behaviour of *ERS-SS* is described, the evaluation of this approach is exposed (*ERS-SS-DAG*). As well as in previous studies (Sections 6.3 and 6.4), we evaluate the model by considering the same parameters' configuration established in scenarios 1 and 2 (Tables 2 and 3). Also the

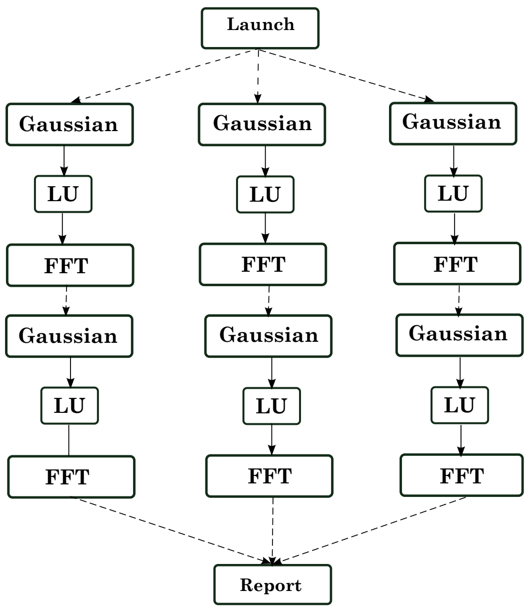


Figure 15: Data flow graph applied in the *ERS* model. Solid arrows imply data and control flow. Dashed arrows imply control flow.

standard European grid technique for selecting resources (*TRS*) executes applications with a *DAG-based* workflow (denoted as *TRS-DAG*).

Figure 16 represents the results for *ERS-SS-DAG* and *TRS-DAG* with Table 2 configuration. Notice that in most of the tests the *ERS-SS-DAG* obtains better execution times with respect to the standard grid selection in European infrastructures *TRS-DAG*. Only with a P_α size of 40 both versions gets similar values.

Next, the data obtained during the second scenario are exposed (Figure 17). The results show that in the early tests (size of J of 50, 100 and 200) each version gets similar execution time values. That is to say, *ERS-SS-DAG* gets values within the range (100, 120) minutes in that three tests while *TRS-DAG* obtains execution time values within the range (155, 180) minutes in the same tests. However, from the size of 300 until the last test (size of 500 tasks) this difference between versions grows continually. In conclusion, the larger the number of tasks to be executed, the greater the benefits of *ERS-SS-DAG*.

Finally, we can say that the proposed model is a feasible solution for *non-DAG* and *DAG-based* workflows in grid environments.

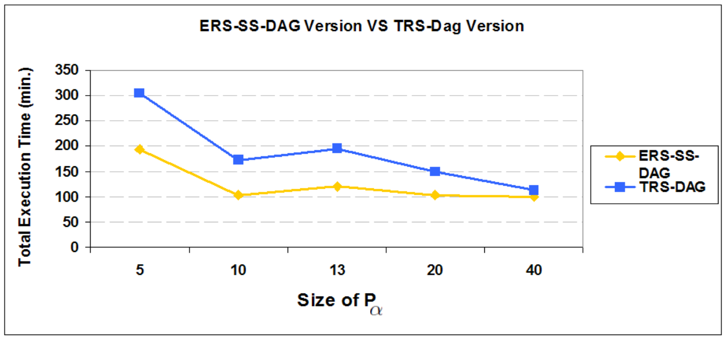


Figure 16: Results obtained with the DAG versions with scenario 1 characteristics. *ERS-SS-DAG* gets better execution time values.

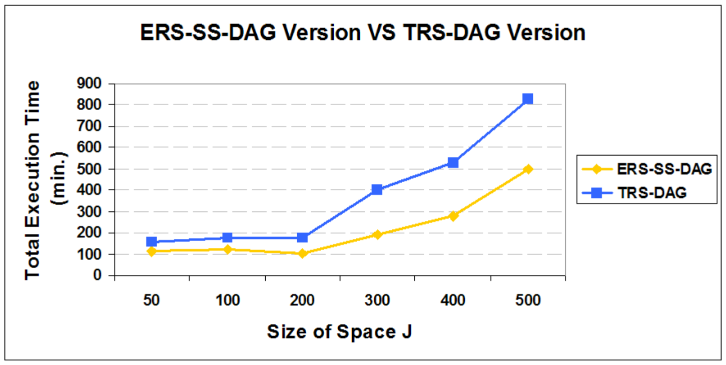


Figure 17: Results obtained with the DAG versions with scenario 2 characteristics. *ERS-SS-DAG* gets better execution time values.

6.6. Concluding Remarks from the Experimental Evaluation

Next, the main conclusions of previous sections (from Section 6.1 to Section 6.5) are summarized:

- In Section 6.1 we have verified that the model learning gets better results when large sizes of initial P_α are applied. More resources are initially evaluated in these cases.
- In Section 6.2 it is possible to observe that the model is a beneficial approach for performing massive computing applications (which are typical samples of grid applications).

- Results in both scenarios confirm that the objectives fixed were accomplished: a reduction in the application's execution time and an increment in the successfully finished tasks rate.

- It is possible to apply task replication in the *ERS* model as shown in Section 6.3 because this option does not affect in a negative way the objectives established.

- When considering free slots in the resources selection, during Section 6.4, we have concluded that efficient resources with a reduced number of free slots (i.e. they have few idle CPUs) handle their tasks queue faster than resources with a lower efficiency value and a higher number of free slots. Therefore, the conclusion is that we should exploit the efficient resources as much as possible.

- Results in Section 6.5 demonstrate that the model manages *DAG-based* workflows as efficiently as *non-DAG* workflows.

- As shown in the experiments along the scenarios discussed, the *TRS* does not outperform the *ERS* in any case. We consider that these facts indicate that we will obtain similar results in other possible experiments.

- Concerning the sensitivity of *TRS*, when using large sizes of P_α more *CEs* are needed at first. In the *TRS* case, that means a higher probability of selecting inefficient resources, leading to a worsening application's execution time. Presumably, in some of those cases *TRS* waits for tasks timeout due to overloaded resources.

- Finally, as said, the *ERS-SS* gets good results for different types of applications as exposed in the diverse scenarios of the present manuscript. Also in these cases different kinds of tasks were taken into account: *Runge-Kutta*, *Gaussian*, *LU* decomposition, and *FFT* algorithm. In addition, we have evaluated the *ERS-SS* approach in other grids, also obtaining good results.

7. Conclusions and Future Work

The present study investigates the problem of resources selection and the application's adaptation in grid environments. This paper proposes an approach that provides this self-adaptive capability to applications increasing their throughput. The model is based only on the operations that users can perform, without modifying of any grid component (hardware or software).

We evaluated our approach in an European grid infrastructure; our efficient selection model has been compared with the standard selection process of European grid environments, which use the gLite middleware. We have also analysed certain characteristics of our model and we have also considered some grid circumstances; all of these issues have been discussed in Sections 6.3, 6.4 and 6.5 within the evaluation phase.

From the results that have been obtained it is possible to deduce that using this model for computing the scientific applications provides, even, a reduction of execution times while improves the number of completed tasks. Another conclusion is that the model can be used as a status indicator of a particular grid infrastructure. That is to say, files with the statistics that the model built after each application's execution can be used as status reports.

Future work will involve improving the model by searching other algorithms that could be better adapted to the grid infrastructure behaviour due to its algorithmic design. Further research is also needed to consider other circumstances or elements that could affect the applications performance. We also aim to use new grid services and facilities to lead in a more powerful and complete self-adaptive capability.

Acknowledgment

María Botón-Fernández is supported by the PhD research grant of the Spanish Ministry of Science and Innovation at the Research Centre for Energy, Environment and Technology (CIEMAT). The authors would also like to acknowledge the support of the European Funds for Regional Development.

References

[1] I. Foster, C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publisher Inc, San Francisco, CA, USA, 2003.

[2] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid. Enabling Scalable Virtual Organizations*, in: R. Sakellariou, J.A. Keane,

J.R. Gurd, L. Freeman (Eds.), LNCS, Euro-Par 2001, Vol. 2150/2001, Springer-Verlag Heidelberg, 2001, pp. 1-4.

[3] N. Mladenovic and P. Hansen. *Variable Neighbourhood Search*, Computers & Operations Research, Vol. 24, Issue 11, 1997, pp. 1097-1100.

[4] P. Hansen and N. Mladenovic. *Variable Neighbourhood Search: Principles and applications*, European Journal of Operational Research Vol. 130, Issue 3, 2001, pp. 449-467.

[5] X.S. Yang and Y.Z.L. Yang, *Cellular Automata Networks*, Proceedings of Unconventional Computing, A. Adamatzky, L. Bull, B. De Lacy, S. Stepney, C. Teuscher (eds.), Luniver Press, pp. 280-302, 2007.

[6] A-L. Barabási and R. Albert, *Emergence of Scaling in Random Networks*, Science, AAAS, Vol. 286, No. 5439, pp. 509-512, 1999.

[7] A. Santiago and R.M. Benito, *An Extended Formalism for Preferential Attachment in Heterogeneous Complex Networks*, Europhysics Letters, EPLA, Vol. 82, Issue 5, 2008.

[8] M. Laguna and R. Martí, *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers, 2003.

[9] M. Laguna and R. Martí, *Scatter Search*, in: E. Alba, R. Mart (Eds.), Metaheuristic Procedures for Training Neural Networks, Springer, 2006, pp. 139-152.

[10] M. Resende, C. Ribeiro, F. Glover, R. Martí, *Scatter Search and Path Relinking: Fundamentals, Advances and Applications*, in: M. Gendreau and J.Y. Potvin (Eds.), Handbook of Metaheuristics, Springer, 2009.

[11] M. Botón-Fernández, F. Prieto Castrillo and M.A. Vega-Rodríguez, *Self-adaptive Deployment of Parametric Sweep Applications through a Complex Networks Perspective*, in: Computational Science and Its Applications, LNCS, Vol. 6783/2011, Springer-Verlag, Berlin Heidelberg, Germany, 2011, pag:475-489. ISBN:978-3-642-21886-6.

[12] M. Botón-Fernández, F. Prieto Castrillo and M.A. Vega-Rodríguez, *Nature-Inspired Algorithms Applied to an Efficient and Self-Adaptive*

Resources Selection Model for Grid Applications, in: Theory and Practice of Natural Computing, LNCS, Vol. 7505, Springer-Verlag, Berlin Heidelberg, Germany, 2012, pag:84-96. ISBN:978-3-642-33859-5.

[13] G. Wrzesinska, J. Maasen and H.E. Bal, *Self-adaptive Applications on the Grid*, 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Jose, California, USA, pp. 121-129, 2007.

[14] J. Buisson, F. Andre, J-L. Pazat, *Supporting Adaptable Applications in Grid Resource Management Systems*, GRID '07 Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Pages 58-65, IEEE Computer Society Washington, DC, USA, ISBN: 978-1-4244-1559-5, 2007

[15] D. Groen, S. Harfst and S. Portegies Zwart, *On the Origin of Grid Species: The Living Application*, LNCS Vol. 5544, pp 205-212, 2009.

[16] H.A. Sanjay and S.S. Vadhiyar, *Strategies for Rescheduling Tightly-Coupled Parallel Applications in Multi-Cluster Grids*, Journal of Grid Computing, Vol. 9, Issue 3, pp. 379-403, 2011.

[17] S.S. Murugavel, S.S. Vadhiyar and R.S. Nanjundiah, *Adaptive Executions of Multi-Physics Coupled Applications on Batch Grids*, Journal of Grid Computing, Vol. 9 Issue 4, pp. 455-478, 2011.

[18] D. Cameron, A. Gholam, D. Karpenko and A. Konstantinov, *Adaptive Data Management in the ARC Grid Middleware*, Journal of Physics: Conference Series, vol. 331, Part 6: Grid and cloud Middleware, 2011.

[19] S.S. Vadhiyar and J.J. Dongarra, *Self Adaptivity in Grid Computing*, in: Concurrency and Computation: Practice & Experience, Vol. 17, Issue 2-4, John Wiley and Sons Ltd., Chichester, UK, pp. 235-257, 2005.

[20] E. Huedo, R.S. Montero and I.M. Llorente, *A Framework for Adaptive Execution in Grids*, Software-Practice & Experience, Vol. 34, Issue 7, John Wiley and Sons Inc., New York, NY, USA, pp. 631-651, 2004.

[21] H.N.L.C Keung, J.R.D. Dyson, S.A. Jarvis and G.R. Nudd, *Self-adaptive and Self-optimising Resource Monitoring for Dynamic Grid Environments*, DEXA'04 Proceedings of the Database and Expert Systems

Applications, 15th International Workshop, IEEE Computer Society, Zaragoza, Spain, pp. 689-693, 2004.

[22] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su and D. Zagorodnov *Adaptive Computing on the Grid Using AppLeS*. IEEE Transactions on Parallel and Distributed Systems, Vol. 14, Issue 4, pp. 369-382, 2003.

[23] D.M. Batista and L.S. da Fonseca, *A Survey of Self-adaptive Grids*, IEEE Communications Magazine, Vol. 48, Issue 7, IEEE Press Piscataway, NJ, USA, pp. 94-100, 2010.

[24] S. Burke, S. Campana, E. Lanciotti, M. Litmaath, P. Méndez Lorenzo, V. Miccio, C. Nater, R. Santinelli, A. Sciabá, *GLite 3.2 User Guide. Manual Series*, CERN-LCG-GDEIS-722398, 2012.

[25] H. Zenil, *Compression-based Investigation of the Dynamical Properties of Cellular Automata and Other Systems*, Journal of Complex Systems, Vol. 19, Issue 1, pp. 1-28, 2010.

[26] J. Yu and R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, Newsletter ACM SIGMOD Record, Vol. 34, Issue 3, pp. 44-49, 2005.