CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–22
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Intelligent Self-adaptive Resources Selection for Grid Applications

María Botón-Fernández[1]* Miguel A. Vega-Rodríguez[2] and Francisco Prieto Castrillo[1]

[1] *Department of Science and Technology, Ceta-Ciemat, Trujillo, Spain.*
Home Page: http://www.ceta-ciemat.es/
[2]*Dept. Technologies of Computers and Communications, University of Extremadura, Cáceres, Spain.*
Home Page: http://arco.unex.es

SUMMARY

Grid computing is considered a promising trend which enables the sharing of a wide variety of computational and storage resources geographically distributed. Despite the advantages of such paradigm, several problems have emerged during the last decade; most of them caused by an inefficient utilization of grid resources.
The present contribution proposes an approach to improve the grid resources selection process. An optimization model for choosing grid resources in an intelligent way has been designed. A mathematical formulation to monitor the resources efficiency has also been established. Furthermore, the model provides a self-adaptive capability to grid applications, enhancing them for dealing with the changing environmental conditions. The model applies an Artificial Intelligence (*AI*) algorithm for ensuring an efficient selection. In particular, three different versions have been implemented. Each of them uses a different *AI* algorithm.
Finally, during the evaluation phase of the model, the experimental tests were performed in a real grid infrastructure. The results show that the model improves the infrastructure throughput, by increasing the finished tasks rate and by reducing the applications execution time.

Received . . .

KEY WORDS: Optimization; Self-adaptive Application; Grid Computing; Evolutionary Computing

1. INTRODUCTION

Around the 50's emerged the idea of using simulated evolution algorithms to solve engineering problems [1] (algorithms are based on the principle of *survival of the fittest*). Then, during the 90's this area was so-called *Evolutionary Computing* (*EC*). Also in this decade appeared the term *Grid Computing* [2]-[4] for denoting a novel distributed system. This type of system enables the sharing and aggregation of a large amount of heterogeneous and geographically dispersed resources. The essential idea was to create a global provider network of computing power and storage capacity, similar to the electrical grid.

In spite of the advantages of this new paradigm, there are several problems related to tasks scheduling, resources selection, resources discovery, resources monitoring, etc. Grid applications compete for using different non-dedicated resources; but also these applications face a double heterogeneity in grid systems: on the one hand, resources are grouped into classes according to their functions and services. On the other hand, every resource class is composed of heterogeneous components, because they are provided by different centres. It must be highlighted that most of

*Correspondence to: Dept. of Science and Technology, Ceta-Ciemat, Trujillo, Spain. E-mail: maria.boton@externos.ciemat.es

these resources are shared among multiple users and projects (non-dedicated), leading to both performance and budget estimation pitfalls. What is more, in a grid environment each resource provider is responsible for managing its own elements (no centralized control). Thus, the availability and characteristics of resources vary over time, making them dynamic and unpredictable [5].

Today, applications require real-time information about grid infrastructures status during their execution. This way, they can deal with the environmental changes. The *adaptation* concept has become a widely used solution in grid community. In recent years, several solutions based on this idea have been proposed [5]-[12]. However, considering the grid characteristics discussed above, the use of adaptation at any grid level has also become a challenge for the scientific community. In this regard, none of the proposed solutions have been placed as a standard across grid platforms.

The present contribution is focused on solving the resources selection problem by applying *adaptation* during the application execution. The proposed approach chooses those resources which best fit the application requirements during execution, without modifying or controlling grid elements. For that reason, a mathematical formulation to obtain the resources efficiency in a grid infrastructure has been proposed. This mathematical formulation combined with an Artificial Intelligent (*AI*) technique provides an Efficient Resources Selection (*ERS*) model. The selection process included in the proposed model is based on *EC* methods. Two main objectives have been established: a reduction in application execution time and an improvement in the successfully finished tasks rate. Three different versions of this model are presented: first, the Preferential Attachment (*PA*) [19] algorithm is adapted to the *ERS* process resulting in the *Efficient Resources Selection Model based on PA* (*ERS-PA*). *PA* is a network evolution model introduced by Barabási and Albert for analysing and understanding random scale-free networks. Second, a metaheuristics algorithm used for solving optimization problems, the *Variable Neighbourhood Search* (VNS) [21], is chosen in the *ERS-VNS* (*Efficient Resources Selection Model based on VNS*). Finally, a Cellular Automata (*CA*) [22] methodology is applied to achieve an intelligent selection by considering geographical criteria (named *Efficient Resources Selection Model based on CA* (*ERS-CA*)). In particular, this latter approach is based on the Conway's Game of Life [23] but also includes ideas from CA networks [24].

Regarding the evaluation phase, two scenarios have been defined. In both of them, the different *ERS* versions are compared with the standard selection used in European grid infrastructures. This way it is possible to determine if the objectives mentioned before are fulfilled (see Section 4). It should be noticed that a real grid infrastructure [25], belonging to an European grid platform, is used to perform the experiments in both scenarios.

The rest of the paper is structured as follows. Section 2 summarizes related work using self-adaptive strategies. In Section 3 a description of the intelligent resources selection model is presented; the three developed versions are also detailed. Section 4 contains all the model evaluation information and the resulting data. Finally, Section 5 concludes the paper addressing future works.

2. RELATED WORK

Nowadays, distributed applications are getting harder to develop, to configure and to maintain. In grid systems this fact is a consequence of the dynamic availability of resources, the infrastructure's heterogeneity, the characteristics of computing nodes, etc. A solution to these problems is to build self-adaptive applications, which are able to explore the computational resources functionality and choose an appropriate resource set in order to maintain a favourable performance.

There are several adaptive approaches focussed on solving the discovery, selection and monitoring processes in grid computing. Some of these works investigate the possibility of providing a self-adaptive capability to grid applications [5] [6]. The main idea is to collect information about resources communication and processing times during applications execution. By using this information, resources are replaced when they reduce the application performance significantly. The study in [5] also tries to solve bottleneck fails.

AppLeS (*Application Level Scheduling Methodology*) [7] is a project that aims to provide an adaptive capability to grid environments. It is considered a novel methodology for adaptive

DOI: 10.1002/cpe

application scheduling by allowing the deployment of adaptive distributed applications. Two main objectives were defined: studying adaptive scheduling for grid computing and applying these results to some applications for verifying their approach. The authors also present various AppLeS enabled applications.

The work in [8] defines a software system which - based on resources characteristics - adapts dynamically its tasks scheduling decisions to the parallelism of the application. Two rescheduling policies, for migration and suspension, are also described in this research. In [9] a new Globus based framework is proposed, allowing users to handle their jobs more efficiently. Other work which uses the adaptivity concept in grid systems is presented in [10]; the contribution describes an approach focused on avoiding the Information System (IS) overload. Moreover, two adaptive notification algorithms are proposed: a sink-based algorithm and an utilization-based algorithm.

The research in [11] proposes two models for predicting the completion time of jobs in a Grid. Both models are used to schedule jobs in two ways: application-level scheduling and system-level scheduling. In [12] a survey of several adaptive grid systems is exposed. A comparison between the described systems is also included.

The work in [13] is focused on designing efficient schedulers in grid environments. They identify different types of scheduling and propose to use heuristic and metaheuristic techniques for enhancing them. This new scheduling process would be more suitable than the traditional ones. In [14] several computational models are described for reaching an efficient grid scheduling. These models use heuristics and metaheuristics. Some of these models use estimation of the computational load of each task, the computing capacity of each resource and an estimation of the prior load of the resources. Other models are based on *TPCC* (Total Processor Cycle Consumption) and also they take into account that resources could change their computing speed over time. There are more realistic models in which the schedulers make decisions based on *GIS* (Grid Information System) about tasks and resources. Finally, in other models the system performance and certain optimization criteria are considered.

Several workflows scheduling algorithms are exposed in [15]. All of them have been developed and deployed in grid environments. In this regard, the study in [16] proposes a taxonomy to characterize several approaches aimed to create and to execute workflows in grid infrastructures. The authors also analyse different workflows systems for a better understanding of the taxonomy. Concerning these contributions, in [17] it is described a taxonomy to classify different software components and high-level methods that are required for autonomic management of applications in Grids. The work also presents a survey about several representative Grid computing systems. One of the most well-known workflows management systems is Pegasus [18], which allows managing abstract workflows that can be executed in different environments without altering the application design.

All the studies mentioned above use scheduling or migration techniques, load balancing, notification policies, etc. That is to say, all these works rely on dynamic resource information. However, the *ERS* model presented in this contribution allows applications to adapt themselves to the changing environmental conditions without modifying or controlling grid resources, without applying migration operations, and without relying on dynamic resource information. This is one of the main differences of our model with respect to the referred works (see Table I). Another difference is that when the model guides applications during their execution, nothing but certain generic application information is needed. This information typically covers the amount of tasks that will be performed, the input and output data, the size of the resulting data, etc.

Finally, Figure 1 shows more information about our approach characteristics and design. This taxonomy is based on the one exposed in [16].

In the following sections the *ERS* model along with the *AI* algorithms that have been performed are described.

DOI: 10.1002/cpe

Table I. Main differences between the related work and the proposed *ERS* model. Please, observe that *ERS* does not use dynamic information of the grid infrastructure.

Solution	New Notification Policies	Dynamic Resource Information
Wrzesinska et al. [5]	-	X
Groen et al. [6]	-	X
Berman et al. [7]	-	X
Vadhiyar et al. [8]	-	X
Huedo et al. [9]	-	X
Keung et al. [10]	X	X
Gao et al. [11]	X	X
Xhafa et al. [13]	X	X
Xhafa et al. [14]	X	X
Yu et al. [15]	X	X
Yu et al. [16]	X	X
Rahman et al. [17]	X	X
Pegasus WMS [18]	X	X
ERS	-	-

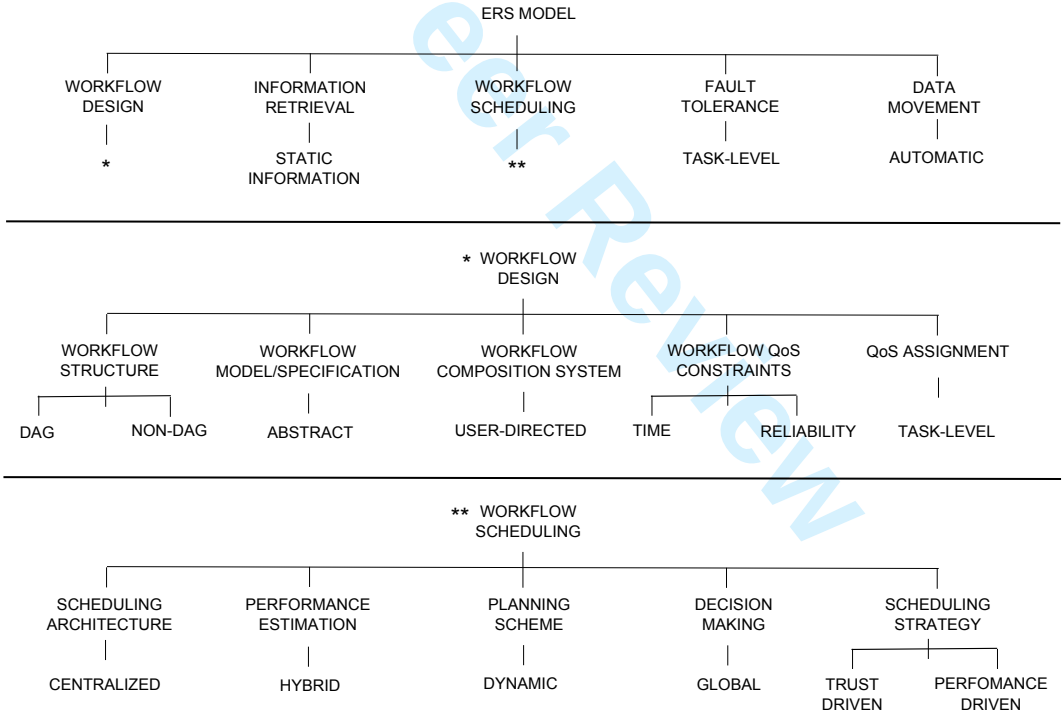

Figure 1. Taxonomy for characterizing and classifying our proposed *ERS* Model in the context of Grid Computing (taxonomy based on [16]).

3. MODELLING AN INTELLIGENT RESOURCES SELECTION SOLUTION

Before exposing the model characteristics and its formulation, certain grid concepts and elements are specified for a better understanding.

3.1. Basic Grid Concepts

Grid computing systems are characterized by an heterogeneous nature and a hierarchical infrastructure. As shown in Figure 2, grid users interact with the infrastructure through the User Interface (*UI*), a machine where users have a personal account. Jobs are submitted through the *UI*. Then, the metascheduler - denoted as Resource Broker (*RB*) - manages these tasks and decides the best site to send them. The *RB* makes decisions about task-resource matching considering tasks requirements. Finally, in every site, there is a scheduler called Computing Element (*CE*) which selects a Worker Node (*WN*) for executing tasks.

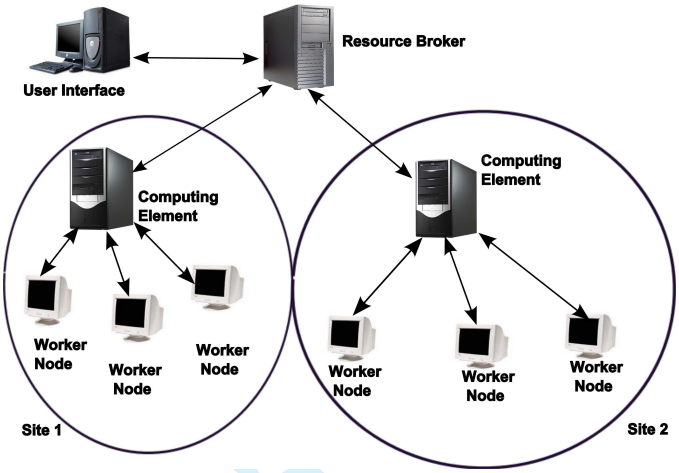

Figure 2. Hierarchical representation of the grid computing elements that operate in the tasks execution process.

The proposed model measures the efficiency of the *CE* schedulers. As stated, for monitoring their efficiency values neither management operations nor scheduling techniques have been performed. In the next section, the mathematical formulation aimed to gauge this efficiency, along with other considerations, is presented.

3.2. Background and Assumptions

As we are intended to guide applications during their execution in grid environments by using the most efficient resources, two spaces are handled by the *ERS* model:

- A task space J consisting of n independent and parallel tasks belonging to the current application (they only differ from input parameter values). The model has been designed for parametric sweep applications.
- A dynamic and heterogeneous resource space R, which has the m available grid resources.

During the application execution, the model is expected to acquire knowledge about the infrastructure in a progressive way. For that reason, the n tasks are not launched at the beginning of the execution but gradually, evaluating and learning at each step. Therefore, the space J is divided into subsets with the same size denoted as P_α. The way to proceed will be by sending a P_α at the beginning of the execution; when this P_α ends its execution the corresponding resources are evaluated. Then, the model efficiently selects resources for a new P_α[†] (see Figure 3). These steps are repeated until all the P_α sets are processed.

The resources selected for a particular P_α compose a RP_α set $\subset R$. Notice that several tasks belonging to P_α could be assigned to the same $CE \in RP_\alpha$ (a many-to-one relationship). For the first P_α launched into execution the RP_α is chosen in a uniform random way because there are no

[†]The different P_α sets are not overlapped in the application execution.

DOI: 10.1002/cpe

efficiency metrics available yet. The remaining RP_α sets are chosen by applying the corresponding *AI* algorithm. The set of processes from sending P_α into execution to the efficient selection of a new RP_α is referred as *model iteration* ($c = 0, c = 1$, etc. are different model iterations).

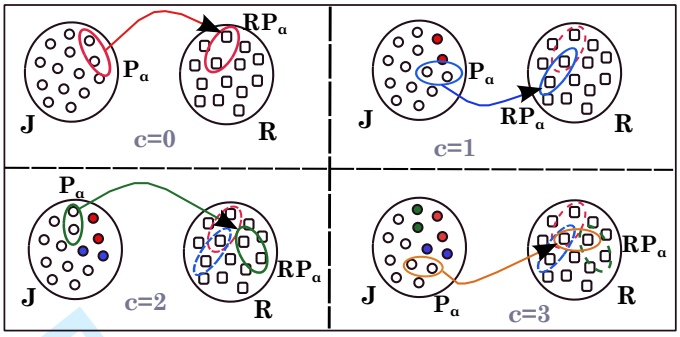

Figure 3. Representation of the mapping between the two spaces handled by the *ERS* model. The processed tasks are coloured and old RP_α sets are batchwise underlined.

The model execution flow (Figure 4) starts when the infrastructure resources are discovered (space R) and the application tasks are determined (space J). Then, tasks are grouped into the corresponding P_α sets; each P_α is built in a random way. Next, the first RP_α is randomly chosen. After that, the P_α is launched into execution and monitored by the *ERS* strategy. It should be noted that each RP_α has associated a *lifetime* lt to complete the tasks belonging to the current P_α. This concept of *lifetime* was introduced to ensure that if a resource collapses, the rest do not wait for it indefinitely. Therefore, once the P_α execution ends[‡], the efficiency values for RP_α resources are measured.

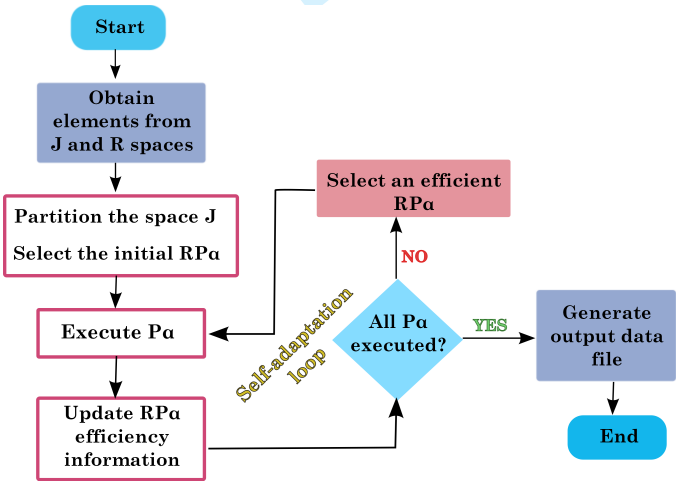

Figure 4. Execution cycle for the proposed *ERS* model. It is possible to observe the loop that provide a self-adaptive capability to environmental changes.

By evaluating the resources efficiency in every *model iteration* an adjustment to the resource set used is allowed as well as a self-adaptive ability to grid application is provided. As mentioned before, these *iterations* imply a model learning, so that, it is expected a better resources specialization when such learning is increased (more *model iterations* imply a better learning).

[‡]The model considers that a P_α ends its execution when either of the following two options occurs: all its tasks are completed or the lifetime runs out.

DOI: 10.1002/cpe

For measuring the efficiency of grid resources a mathematical formulation is fixed. The fitness value for the ith CE depends on two parameters (Eq. 1): on the one hand, the percentage of successfully completed tasks ϵ_i and, on the other hand, the normalized increment ΔT_i of its processing time (Eq. 2).

$$F_i = (a \cdot \epsilon_i + b \cdot \Delta T_i)/(a + b)\,. \tag{1}$$

Both parameters are obtained for every *CE* selected during the application execution. The values of parameters a and b are introduced by users and indicate the relevance of ϵ_i and ΔT_i, respectively. Thus, users can specify the priority conditions of their experiments. The parameter ΔT_i is calculated considering the maximum and minimum processing time values (denoted as T_{max} and T_{min} respectively) within the corresponding RP_α. It could be considered as a time ranking within every RP_α subset.

$$\Delta T_i = (T_{max} - T_i)/(T_{max} - T_{min})\,. \tag{2}$$

Finally, the processing time T_i is defined in Eq. 3. NT_i is the set of assigned tasks to resource ith, $Tcomm_i$ is the communication time between resource i and other grid services and $Tcomp_{j,i}$ is the processing time acquired by resource i when executing task j.

$$T_i = Tcomm_i + \sum_{j \in NT_i} Tcomp_{j,i}\,. \tag{3}$$

As stated, the mathematical formulation is combined with an *AI* algorithm, resulting in a *ERS* model. Figure 5 represents the different steps performed by this combination. Steps $1-4$ form the initialization phase, in which the spaces are determined and the first P_α is launched into execution. The model performs step 5 for every P_α, so that, each time this step is repeated a new *model iteration* is considered. Furthermore, step 5.3 includes the particular *AI* rules which are described in the following subsections (these rules are applied to all resources $\in RP_\alpha$). Specifically, we have developed three *ERS* versions: the first one is based on Preferential Attachment (*ERS-PA*) [26], the second one is based on the Variable Neighbourhood Search (*ERS-VNS*) and the third one is based on Cellular Automata (*ERS-CA*).

ERS PSEUDOCODE

Input: application tasks, infrastructure resources

Output: Efficient resources

1. Set J and R spaces;
2. Divide J in P_α sets;
3. Compose the first RP_α randomly;
4. Launch the first P_α into execution;
5. While there are unprocessed P_α **do**
  5.1 Monitor P_α;
  5.2 When P_α finishes: update efficiency metrics for RP_α resources;
  5.3 Apply AI algorithm for efficient selection of new RP_α;
  5.4 If there are unprocessed P_α **then** Launch a new one;
6. End while

Figure 5. The main steps of the resulting *ERS* model are summarized. Step 5.3 varies according to the algorithm applied.

DOI: 10.1002/cpe

3.3. The PA Algorithm for the Efficient Resources Selection

The preferential attachment technique [19] is a characteristic algorithm from the complex systems field in which nodes are continuously added to the network. The probability of linking a new node j with an existing one i in the complex network is proportional to the node's degree k_i. Therefore, those nodes with higher degree (*hubs*) have more probability of getting a new link. For example, in a social domain the most popular people would be more likely to make new friends. The following assumptions are fixed in *ERS-PA*:

- Elements belonging to space R are considered as nodes of a complex network (see Figure 6), which is generated at runtime.
- Hubs in this network represent the most efficient resources, i.e. those resources which are most often used to perform tasks from different P_α sets.
- Thus, the degree of a node (resource) indicates the number of times this *CE* has been selected during execution (establishing a resource-task connection).
- Network edges represent the constraint *"executing tasks from the same* P_α*"*, which is equivalent to *"resources belonging to the same* RP_α*"*.
- Every RP_α composes a complete subgraph within the complex network.

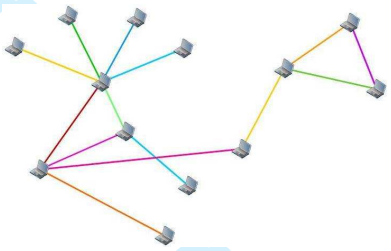

Figure 6. The *ERS-PA* version builds a complex network using the selected infrastructure resources at runtime. *Hubs represent the most efficient resources*

There is a *PA* extension, denoted as Heterogeneous Preferential Attachment (*HPA*) [27], in which additional characteristics are considered during the calculation of the link probability. In *ERS-PA*, the link probability is based on both node degree and resource fitness (Eq. 1). Moreover, both parameters determine the resource efficiency as shown in Eq. 4.

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

Parameter k_{max} is the maximum degree value within the evaluated RP_α; k_i and F_i are the degree and fitness values of the ith resource. These two parameters are calculated for all RP_α resources.

Finally, the scheme with the main selection rules of *ERS-PA* is presented in Figure 7 (extending point 5.3 of Figure 5). The efficiency values gathered at that moment are used to determine the link probability. Whenever a RP_α is fixed, the complex network is updated.

3.4. The Efficient Resources Selection from the VNS Perspective

The *Variable Neighbourhood Search* VNS [21] is a metaheuristic used in global optimization problems. It is based on changing the environment structure when the local search stagnates (see Figure 8). The main idea is to maximize a particular function $max\{f(x)|x \in X\}$ where x is a feasible solution, f is the objective function and X is the space of feasible solutions for the problem.

Furthermore, an structure of environments within the solution space X is considered an application $N : X \rightarrow 2^X$, that associates to each solution $x \in X$ an environment of solutions $N(x) \subset X$. Denoting the finite set of environment structures as $N_h(h = 1, ..., h_{max})$ and the set of solutions for the h^{th} environment as $N_h(x)$, the rules of *VNS* are as follows:

- **Initialization:** select the set of environment structures N_h; specify an initial solution x and choose a stopping condition.

DOI: 10.1002/cpe

ERS-PA SELECTION PSEUDOCODE

5.3 Efficient resources selection for a new RP_α using PA;
5.3.1 Obtain the maximum efficiency value labelled as E_{max};
5.3.2 Within the range (0, E_{max}) an efficiency value E_r is randomly chosen;
5.3.3 A resource is randomly selected from R and its efficiency E_n is compared with E_r;
5.3.4 If $E_n >= E_r$ **then** the resource is added to the new RP_α set;
otherwise is neglected;

Figure 7. The *ERS-PA* pseudocode including the main selection actions. It is possible to observe how the link probability is calculated.

- **Repeat until the stop condition:**
  - set $h = 1$
  - Repeat until $h = h_{max}$:
    - **Shaking:** generate a random solution x' from the h^{th} environment of x.
    - **Local search:** apply some local search technique using x' as initial solution. Denote as x'' the local optimum which has been obtained.
    - **Move or not:** if this local optimum is better than x, move there ($x = x''$ and $h = 1$); otherwise set $h = h + 1$.

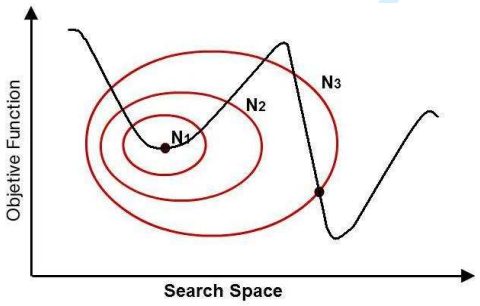

Figure 8. The *VNS* algorithm avoids stagnations in the search of solutions by changing the environment during the process.

In the *ERS-VNS* model, it is assumed that every RP_α is a solution within the search space. So that, we are looking for the optimum RP_α set. For that reason, the fitness of the whole RP_α is measured as shown in Eq. 5. It is based on F_i average value. Parameter d is the cardinality of RP_α.

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

The following considerations were defined for this particular *ERS* version:

DOI: 10.1002/cpe

- x is the first RP_α set launched into execution. It is also known as *initial solution*.
- x': is the RP_α set which is obtained by applying a mutation process to x.
- x'': is the new RP_α set acquired after performing the *local search process* to x'. Moreover, it could be the set for the following P_α.

Regarding the finite set of environment structures $N_h(h = 1...h_{max})$, the value of k_{max} is fixed at 5. For determining this value the next factors were taken into account: a significant change of environment which implies finding the optimum solution faster, the available resources in the real infrastructure and the size of the experiments. Furthermore, every neighbourhood has associated two parameters (see Figure 9) :

1. p_k specifies the percentage of variation of a solution x' with respect to the initial solution x when the *mutation process* is performed.
2. q_k specifies the search range during the candidates selection (the number of possible candidates to be part of the solution x''). It is used within the *local search process* for obtaining the solution x''.

Notice that p_k varies in a non-decreasing way from an environment to another because it represents how different is the initial solution x with respect to the solutions of the corresponding environment. On the other hand, q_k is used to specify the size s of the candidate set in every environment. This size is defined as the q_k percentage of g (number of available resources). The way to do that is by ordering the g resources from higher to lower efficiency values. Then the q_k percentage is applied and the candidate set is composed. Finally, the model selects which resources will run new P_α tasks from the best s resources. In order to improve the *VNS* advanced behaviour, it has been established a threshold of goodness U for the resources fitness (the threshold value is obtained through experimental tests). This goodness threshold is used during the *mutation process* to determine what resources should mutate. So that, those resources whose fitness value does not exceed this threshold are considered as inefficient.

ERS-VNS SELECTION PSEUDOCODE

5.3 Efficient selection of RP_α using VNS;
  5.3.1 Set k=1;
  5.3.2 While k <= k_{max} **do:**
    5.3.2.1 Shaking: generate a point x' randomly from the kth neighbourhood of x.;
    5.3.2.2 LocalSearch: apply some local search method using x' as initial solution. Denote with x" the obtained local minimum.;
    5.3.2.3 Move or not: if x" is better than x **then**:
      * x=x" and k=1
      * **Otherwise** k=k+1;

Figure 9. Representation of the algorithmic for the selection process within the *ERS-VNS* version. Solution x'' is equivalent to the new RP_α.

3.5. The Efficient Resources Selection based on the Cellular Automata Methodology

Cellular automatas (CA) [22] are computational models composed of a discrete grid of cells; each cell has a value belonging to a specific set of states. According to some fixed rules, the new state of each cell is determined (considering the current state of the cell and the states of its neighbours).

From the results that have been obtained in both *PA* and *VNS* versions, we consider an important improvement to include geographical criteria in the efficient selection process. That is to say,

DOI: 10.1002/cpe

if a resource belonging to a particular site is down or overloaded, other resources at that site or in a close one may be in a similar situation. In that case, a promising option would be to evaluate resources belonging to different sites. That is the reason why the *CA* methodology is combined with the mathematical formulation, resulting in the *ERS-CA* version. We represent the grid infrastructure as a cellular automata network [24] [28], where *CEs* are the corresponding cells and the topological aspects are also introduced. The proposed *CA* is divided into 8 subnetworks according to geographical proximity criteria, as shown in Figure 10. The number of subnetworks depends on the number of resources available in the testbed infrastructure. This information is specified through a configuration file which can be manipulated by users.

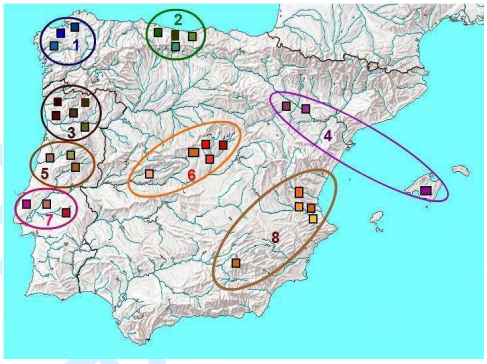

Figure 10. Geographical distribution of resources in IBERGRID project and sub-networks clustering for the *ERS-CA* case.

As stated, every cell in the *ERS-CA* represents a resource whose neighbours in the CA correspond to physical neighbours in the infrastructure. The rules that govern this version (Figure 11) are described below.

Every cell has three possible states: *alive, dead or inactive*. An *alive* cell is a selected resource which is considered as efficient. On the contrary, any resource that is not considered efficient is established as *dead*. Finally, the resources that are inoperative or unavailable have the *inactive* state. The 8 subnetworks are composed at least by three resources (considering the real quantity of resource in every site of the testbed infrastructure). During the *selection process* just two cells survive from each subnetwork: the two less loaded resources (avoiding overload). This way, resources of a given subnetwork compete for survival. The model evaluates the resources workload to make this decision. The rules for governing every subnetwork are as follows:

- Every *alive* cell which is overloaded dies.
- Only the two resources with minimum load are considered living cells.
- Any cell with all its neighbours dead survives.

The surviving resources from all subnetworks constitute a candidate set. The next rules are applied over this candidate set to obtain the new RP_α (i.e. the alive cells would be the resources that finally compose a RP_α).

- **Selective pressure**: only the most efficient 50% promotes to be part of the new RP_α.
- **Scout resource**: a resource is selected in a random way to form the new RP_α. This rule is applied only if there are unexplored resources (that have not been used until now).

4. EXPERIMENTS AND RESULTS

4.1. Test-bed Infrastructure

To evaluate the performance of the *ERS* model, we executed a set of experiments in the Spanish National Grid Initiative (*ES-NGI*) [25], a production distributed computing environment that is part

DOI: 10.1002/cpe

ERS-CA SELECTION PSEUDOCODE

5.3 Efficient selection of RP_α using CA;
  5.3.1 Evaluate RP_α resources workload;
  5.3.2 Candidate set: the two resources less loaded promote in every subnetwork;
  5.3.3 Get alive cells
    5.3.3.1 Selective pressure: only the most efficient resources from the candidate set will be alive cells;
    5.3.3.2 Scout: an unusued resource will be chosen randomly to be an alive cell;
  5.3.4 Update cells status in the CA;
  5.3.5 Compose a new RP_α with the alive cells;

Figure 11. Illustration of rules for the *ERS-CA* version. These steps determine the behaviour of the infrastructure as a CA

of the European Grid Initiative (*EGI*) [29]. The authors are affiliated with the Ibergrid *VO* [30], a virtual organization (*VO*) of *ES-NGI* that contains about 30 *CEs* located between Spain and Portugal. The results of the *ERS* experiments are evaluated with respect to both a reduction of the execution time and an improvement of the number of successfully completed tasks for grid applications.

Related to the testing application, a fourth-order approximation of the Runge-Kutta method [31] has been considered; this is a well-known procedure used to solve systems of ordinary differential equations. As we focus on analysing the *ERS* model behaviour, the application plays the role of a testing component in the evaluation process. Hence, no further considerations about the scientific production are taken into account in the defined scenarios.

Finally, it should be highlighted that we have compared the implemented versions with the standard selection technique used by the infrastructure concerned. In particular, our proposal is compared with *gLite* [32], which is the middleware used in the European grid infrastructures (see projects *EGEE* [33] and *EGI* [29]). In conclusion, it is compared with the scheduling techniques and policies of the current European grid infrastructures. This selection, that we denoted as *TRS* (Traditional Resource Selection), is performed by the *WMS* (Workload Management System) in a process called *match-making*, which selects those available *CEs* that fulfil the user requirements and that are close to input Grid files. Moreover, the resulting data that compose this baseline are the average of several experiments data.

4.2. Scenario 1: Evaluating the Model Learning

As stated, every time step 5 is performed (Figure 5), a new *model iteration* is considered. Moreover, during this step the model learning occurs. The number of times this process is repeated depends on how many P_α are established, namely, it depends on P_α size (all P_α have the same size). Then, we assume that the size of P_α affects the model learning, delaying or advancing it. For that reason, several tests are performed using different values for P_α size. Particularly, the scenario is composed of 5 tests where 10 real experiments are performed for every version (*ERS* versions and *TRS*). So that, every graphical point is the average value of these 10 experiments. The size of the space J is fixed at 200 tasks for all tests. As shown in Table II, the iterations are gradually increased affecting

DOI: 10.1002/cpe

directly the P_α size. The values for parameters a and b are 60% and 40% respectively. This way, we give a slightly higher importance to the number of successfully completed tasks with respect to execution time (it is considered more important to finish as many tasks as possible during the application execution).

Table II. Values of P_α for each test of this scenario along with the number of iterations to be performed.

Test	Space J	P_α size	Iterations
1	200	40	5
2	200	20	10
3	200	13	15
4	200	10	20
5	200	5	40

Observing the results that have been obtained (Figure 12) we can conclude that the intelligent resources selection model improves the execution times with respect to the *TRS*. This improvement is appreciated in the 3 evaluated versions. We discuss below the performance of each implemented version taking as reference the behaviour of *TRS*. The time difference between *ERS-CA* and *TRS* is improved from the first point until the last one. It can be observed that for the last points $(13-40)$ the *ERS-CA* achieves its maximum execution time difference with respect to *TRS*. In these tests the size of P_α is larger, which implies that in every model iteration a larger amount of resources is evaluated. This is due to both the survival and selective pressure rules along with the effect of the P_α size. When a P_α is processed, there are fewer resources to evaluate (small P_α size). Hence, the automata has fewer resources able to survive from one *model iteration* to another (slow training model).

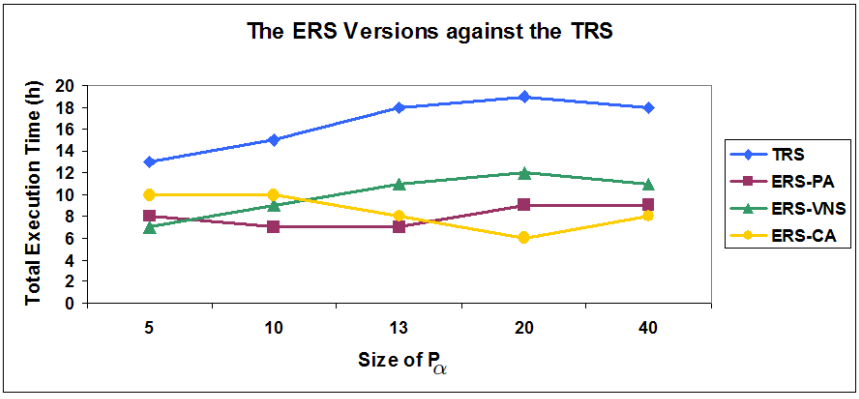

Figure 12. Execution time functions which have been obtained from the tests performed in scenario 1 when evaluating the *ERS* model. Every point is the average value of the simulations performed.

Concerning *ERS-VNS*, as shown in Figure 12, the size of P_α does not affect negatively its performance; the execution time difference with respect to *TRS* is almost constant for all tests. In this case, for lower sizes of P_α *ERS-VNS* gets the minimum execution time values (there are more model iterations). This is due to the fact that every model iteration implies a change of environment, i.e., an approach to the optimal solution (the most efficient resources).

The *ERS-PA* version gets its minimum value at the third test (size of 13), but in the last two tests that difference is reduced. As stated in Section III.C, in this version the efficiency is based on the resources fitness and degree. When an efficient resource gets overloaded, its fitness function gets worse but its degree remains constant and even grows (it represents the number of times a resource is used by the model), so that, its resulting efficiency value decreases slightly. This fact causes that

the model undergoes into several iterations to detect that the resource is overloaded. This leads to a decrease in the *ERS-PA* performance as it can be appreciated in Figure 12.

In Table III[§] some descriptive statistics concerning the total execution time for the three proposed versions of the model (*ERS-PA*, *ERS-VNS*, and *ERS-CA*) are included. It must also be highlighted that the coefficient of variation never exceeds 50% of the mean. Please, notice that the values for the standard deviation are motivated by the dynamic and changing nature of grid infrastructures.

Table III. Statistical values of *ERS-PA, ERS-VNS* and *ERS-CA* in the corresponding tests.

TRS	Size 5	Size 10	Size 13	Size 20	Size 40
Mean	13	14	16	18	17
Standard Deviation	4.2	2.97	5.84	2.22	0.23
Coefficient Variation	32%	21%	37%	12%	1%
ERS-PA	**Size 5**	**Size 10**	**Size 13**	**Size 20**	**Size 40**
Mean	8	7	7	9	9
Standard Deviation	0.8	1.05	0.92	0.7	0.8
Coefficient Variation	10%	15%	13%	8%	9%
ERS-VNS	**Size 5**	**Size 10**	**Size 13**	**Size 20**	**Size 40**
Mean	7	9	11	12	11
Standard Deviation	0.52	0.58	0.75	0.58	0.81
Coefficient Variation	7%	6%	7%	5%	7%
ERS-CA	**Size 5**	**Size 10**	**Size 13**	**Size 20**	**Size 40**
Mean	10	10	8	6	8
Standard Deviation	0.48	0.82	0.8	0.82	0.8
Coefficient Variation	5%	8%	10%	13%	10%

Finally, according to the second goal, an increase in the number of successfully completed tasks is achieved: from a 73% of success rate for *TRS* to a 95.6%, 92.7% and $87,2\%$ in *ERS-VNS*, *ERS-PA* and *ERS-CA* respectively.

4.3. Scenario 2: Evaluating the Effects of J Space Size

In this second scenario we evaluate the model behaviour for large productions. This way we determine the range of applications in which the *ERS* strategy can be applied. As shown in Table IV, in this case 6 tests were performed, in which the size of the J space varies from 50 to 500 tasks. As far as P_α size is concerned, from the first scenario it is concluded that reaching a good training model and evaluating a suitable amount of resources in every *model iteration* are equally important. Thus, P_α has been set at 10, resulting in the next range of model iterations: 5, 10, 20, 30, 40 and 50. As in the preceding scenario, the values assigned to the *relevant parameters* a and b are 60% and 40% respectively. Finally, it should be noted that a better reduction of execution times for all the productions is expected.

From the results we have obtained (Figure 13) it is possible to conclude that the intelligent model improves the execution times in its three versions with regard to the Traditional Selection (*TRS*) of grid infrastructures. For the first three sizes of space J $(50 - 200)$, the results in the *ERS* versions are very closed and also do not show a large difference with respect to *TRS*. However, from the space of 300 tasks this difference starts growing significantly.

Similarly to the first scenario, from a size of 300 tasks the *ERS-PA* increases the execution time. That means that the efficient resources overload and the model has to adjust their efficiency values

[§]In the tables the average is expressed in hours and the other statistics have been calculated based on this measure.

DOI: 10.1002/cpe

Table IV. Set of values for each test performing in scenario 2. In all tests experiments of the developed versions along with *TRS* are executed.

Test	Space J size	P_α size	Iterations
1	50	10	5
2	100	10	10
3	200	10	20
4	300	10	30
5	400	10	40
6	500	10	50

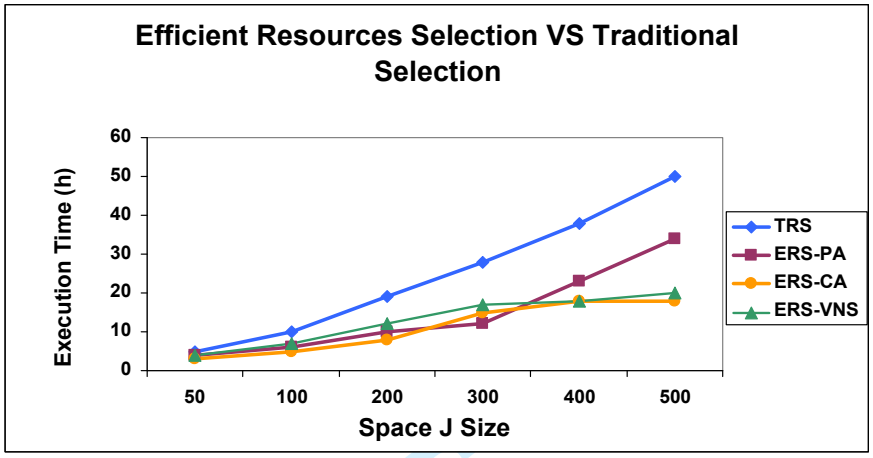

Figure 13. Execution time values obtained during the tests of this second scenario. The efficient versions attain better time reductions against *TRS*.

(this adjustment takes time resulting in a performance detriment). On the other hand, both *ERS-VNS* and *ERS-CA* have a similar behaviour for all tests, progressively improving their execution times (increasing the difference with respect to *TRS*).

Also the corresponding statistical values for the model versions within this second scenario have been included (see Table V).

Finally, it can be concluded that the *ERS* model achieves the proposed goals established during the definition phase; the assumptions related to its behaviour have also been demonstrated. Hence, making use of the intelligent model would result in significant improvements for grid users.

4.4. Scenario 3: Considering DAG (Directed Acyclic Graph) applications

Applications deployed in grid environments can be grouped in two main sub-taxonomies of workflows structures as described in [16]: *DAG*-based workflows and *Non-DAG*-based workflows. These two types of workflows structure are also categorized into several kinds of patterns. In *DAG*-based workflows the *sequence, parallelism* and *choice* patterns are included. An ordered series of tasks, in which a particular task starts its execution after a previous one has completed, is defined as *sequence*. When tasks are executed concurrently their interactions are represented as a *parallelism* pattern. Finally, *choice* it is used when a task is chosen to be executed if its associated conditions have been accomplished. In *Non-DAG*-based workflows are considered the previous patterns along with the *iteration* structure (it represents tasks which are allowed being repeated in an iteration block).

The previous scenarios have been carried out by applying a parametric sweep application (*Non-DAG*-based workflow with a *parallelism* pattern) characterized by independent and parallel tasks (they only vary in the value of input parameters). However, in this scenario an application with a

Table V. Statistical values of the *ERS* versions within the second scenario.

TRS	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	6	10	17	28	39	50
Standard Deviation	0.99	2	2.98	4.53	11.03	11.65
Coefficient Variation	17%	20%	18%	16%	18%	23%
ERS-PA	**Size 50**	**Size 100**	**Size 200**	**Size 300**	**Size 400**	**Size 500**
Mean	4	6	10	12	23	34
Standard Deviation	0.7	1.08	0.75	0.83	0.72	0.8
Coefficient Variation	17%	18%	8%	7%	3%	2%
ERS-VNS	**Size 50**	**Size 100**	**Size 200**	**Size 300**	**Size 400**	**Size 500**
Mean	4	7	12	17	18	20
Standard Deviation	0.63	0.73	0.48	0.32	0.8	0.8
Coefficient Variation	16%	10%	4%	2%	5%	5%
ERS-CA	**Size 50**	**Size 100**	**Size 200**	**Size 300**	**Size 400**	**Size 500**
Mean	4	7	12	17	18	20
Standard Deviation	0.57	0.78	0.65	0.47	0.82	0.8
Coefficient Variation	19%	16%	8%	3%	5%	4%

DAG-based workflow is considered (also with a *parallelism* pattern). The workflow (see Figure 14) is comprised of three well-known algorithms: *Gaussian*, *LU* (Lower/Upper) Decomposition, and *FFT* (Fast Fourier Transform). Then, the different tasks that composed our particular *DAG* workflow are connected by directed arcs and are represented as nodes. There are also other two nodes that do not represent a computational task: *Launch* and *Report*.

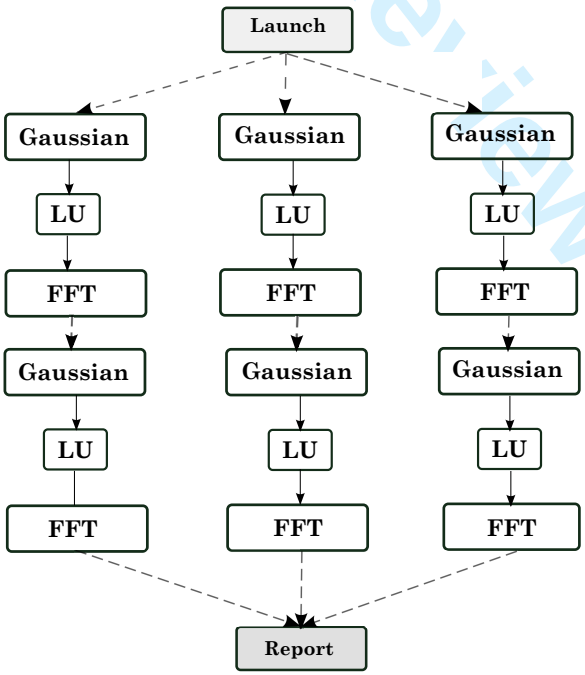

Figure 14. *DAG*-based workflow applied during the third scenario.

DOI: 10.1002/cpe

The *Launch* node sends control directives to nodes with which it is connected (dashed gray arcs) for starting their execution. The remaining nodes receive their input data from other nodes through their input arc(s) according to the established dependencies. Finally, the *Report* node is connected with the last nodes; these nodes are responsible of sending their computed solutions along to their output arc(s). The dependencies between tasks are established in the following manner: *Gaussian-LU-FFT*. Thus, it is possible to observe that *FFT* nodes have two feasible behaviour: **1)** while there are non-executed tasks these nodes initialize the next *Gaussian* nodes. **2)** When all tasks have been performed the *FFT* nodes transfer their solutions to the *Report* node.

In this third scenario the *ERS* model has a similar behaviour as when performing Non-DAG-based workflows. In this regard, at the beginning of the application execution several *Gaussian* tasks are launched (the number of tasks is the same as P_α cardinality). In contrast to *Non-DAG* execution flow, when a task $t_j \in J$ ends its execution the efficiency of the corresponding resource is measured. Then, a new efficient resource is selected for executing a new task. The next rules are considered for determining the type of task to be executed:

- A *Gaussian* task transmits its solution to a *LU* task.
- Every *LU* task transfers its computed solution to a *FFT* task.
- As stated, a *FFT* task initializes a *Gaussian* task or reports its solution to the *Report* node as necessary.

Notice that in *DAG*-based workflows the *lifetime* is assigned to every task due to the fact that the model evaluates resources in an independent way (without considering a RP_α set). Now, these new versions of the model are tested and compared to the standard selection (*TRS*) which also executes a *DAG*-based workflow. Table VI shows the characteristics of this scenario.

Table VI. Set of values for each test performing in scenario 3. These values are similar to the scenario 2 configuration.

Test	Space J size	Initial tasks sent
1	50	10
2	100	10
3	200	10
4	300	10
5	400	10
6	500	10

Figure 15 represents the results of the *ERS* model considering *DAG*-based workflows and applying the three algorithms (the different versions are denoted as *ERS-PA-DAG*, *ERS-VNS-DAG* and *ERS-CA-DAG*, respectively). The three versions reach better execution times with respect the standard selection in European grid infrastructures (*TRS-DAG*). The *ERS-PA-DAG* achieves a higher execution time difference with respect to *TRS-DAG* from the beginning of tests. However, in last test (size of 500) it is possible to observe that its execution time increases, which is probably motivated by a *hubs* overload (as specified in Scenario 1, this model version requires several iterations before detecting that a resource is overloaded). *ERS-VNS-DAG* and *ERS-CA-DAG* attain close values for a size of 50 tasks but then, *ERS-CA-DAG* improves its execution time difference with respect to *TRS-DAG* (from 100 to 500 tasks) and also in regard to *ERS-VNS* for the sizes of 100 and 200 tasks. Meanwhile, *ERS-VNS-DAG* maintains an execution time difference of 20 minutes in the first three tests (sizes of 50, 100, and 200) with respect to *TRS-DAG*. After that, it extends its execution time difference with respect to *TRS-DAG* due to the fact that for higher task sizes the model reaches a better specialization (i.e. values close to the global optimum are accomplished).

Also the corresponding statistical values for the *DAG* versions have been included (see Table VII). As well as in previous sections, the coefficient of variation never surpasses 50% of the mean. In *ERS-VNS-DAG* as the size of J is increased, the coefficient of variation value is reduced.

DOI: 10.1002/cpe

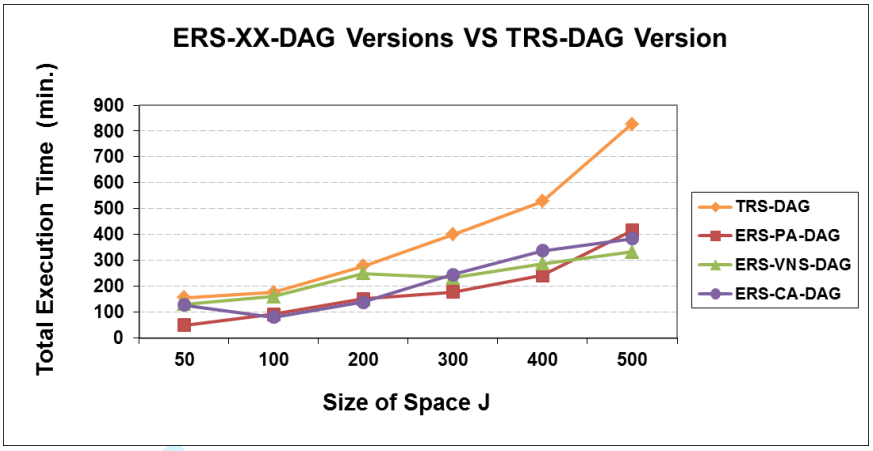

Figure 15. Results obtained for the different *ERS* versions when executing *DAG*-based workflows.

Table VII. Statistical values of the *ERS-DAG*-based versions within the third scenario.

TRS-DAG	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	155	176	278	400	528	827
Standard Deviation	53.21	55.13	52.3	56.31	76.13	80.64
Coefficient Variation	34%	31%	19%	14%	18%	10%
ERS-PA-DAG	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	49	91	151	178	242	416
Standard Deviation	8.38	23.31	35.33	11.78	56.02	46.20
Coefficient Variation	17%	26%	23%	7%	23%	11%
ERS-VNS-DAG	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	131	160	249	232	287	333
Standard Deviation	52.88	45.16	54.87	28.89	12.56	38
Coefficient Variation	40%	28%	22%	12%	4%	11%
ERS-CA-DAG	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	132	81	139	245	337	384
Standard Deviation	61.13	7.66	23.86	26.11	60	34.89
Coefficient Variation	46%	9%	17%	11%	18%	9%

4.5. *Analysing Compute Time and Scalability*

Scalability is a desirable ability of systems, networks and/or applications. Concerning algorithms or programs, this ability shows if they are suitably efficient and practical when they are applied to large situations (by varying some parameters). In this Section we analyse how the proposed versions of *ERS* scale for real-world workloads, by changing the problem size. For that reason, the compute time of our model has been measured considering the sizes of J in the second scenario and some other values (see Table VIII). Thus, it is possible to determine how the model will perform massive computing applications and also to stablish how it penalizes the application execution time in a grid infrastructure. Notice that the value of P_α is fixed at 10, in the same manner as in scenarios 2 and 3.

Figure 16 shows how the three versions scale for the different cases taken into account. Although the three algorithms scale in a similar way in the first tests (from 50 to 500 tasks), *ERS-VNS* has a worse scaling behaviour in the last ones with respect to *ERS-PA* and *ERS-CA*.

DOI: 10.1002/cpe

Table VIII. Compute time obtained for the different *ERS* algorithms considering several sizes of space J. The resulting time is expressed in minutes.

Size of J	Compute Time *ERS-PA*	Compute Time *ERS-VNS*	Compute Time *ERS-CA*
50	0.08	0.07	0.12
100	0.13	0.15	0.2
200	0.33	0.38	0.35
300	0.4	0.75	0.53
400	0.43	1.23	0.67
500	0.65	1.9	0.82
1000	1.33	7.22	1.58
1500	1.85	17.78	3.15
2000	2.47	32	3.52

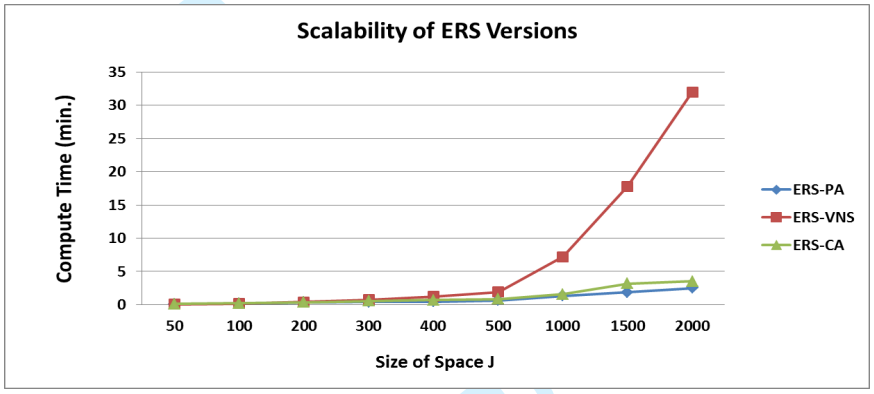

Figure 16. The resulting compute time for the three proposed *ERS* versions. *ERS-VNS* scale in a worse way.

It must also be highlighted that none of them consumed more than 0.5% of the application execution time (taking as reference the data results of the second experiment for the three algorithms). Finally, the best way to decide which algorithm to use would be by considering not only these data but also the results discussed in Sections 4.2, 4.3 and 4.4.

4.6. Analysis of Parameters a and b

Finally, the effect of relevance parameters a and b has been analysed. Furthermore, several pairs of values are fixed for studying how the model behaves in these cases. It is expected a better understanding of the proposed model and also a justification for the pair of value $(60, 40)$ used in the previous scenarios. Table IX includes the different instances which have been considered (they are the most significant pairs of values).

Table IX. Set of values used for parameters a and b.

Test	**Parameter a value**	**Parameter b value**	**Size of J**
1	20	80	200
2	40	60	200
3	60	40	200
4	80	20	200

DOI: 10.1002/cpe

From the results shown in Figure 17 it is possible to observe that when the execution time is prioritized (pairs $(20, 80)$ and $(40, 60)$) the *ERS-VNS* and *ERS-CA* reach their minimum execution time values. In the same way, when the successfully finished task rate gets the higher priority the execution time is penalized. The *ERS-PA* has a similar trend that the other ones with the exception of pair $(60, 40)$, in which it achieves its minimum execution time value. Considering the execution time, the best pair of value to use is $(60, 40)$ but we also analyse the successfully finished task rate to determine which one is the best option.

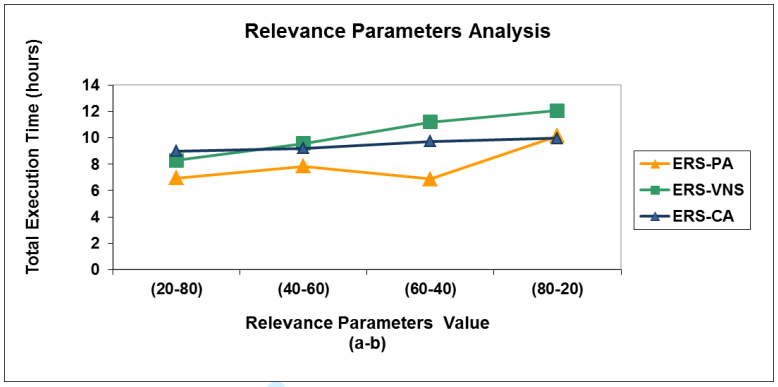

Figure 17. Analysis of parameters a and b in the different *ERS* versions.

Figure 18 includes the successfully finished task rates that the three *ERS* versions have obtained during the analysis of relevance parameters. All the proposed approaches reach the best successfully finished task rate at the point of $(60, 40)$. Finally, we conclude that by using this pair the algorithms acquire a suitable balance between execution time and finished tasks.

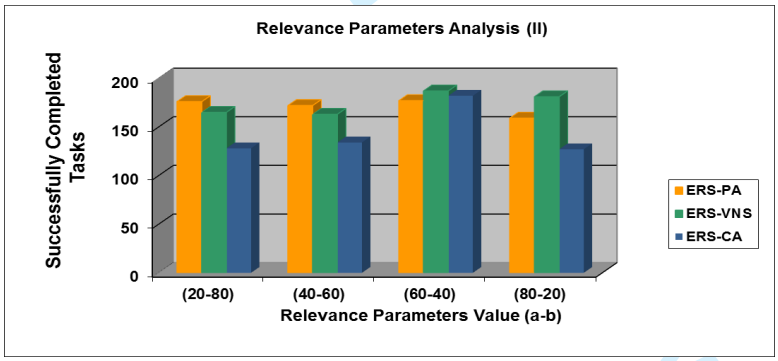

Figure 18. The resulting successfully finished task rate during the analysis of relevance parameters.

5. CONCLUSION

In the present research we focused on solving the resources selection problem, looking for an adaptation deployment of applications in grid environments. This way, applications would be able to deal with the environmental changes. For this purpose, we propose an intelligent selection model which provides a self-adaptive capability to grid applications, increasing system throughput. Furthermore, the model (denoted as *Efficient Resources Selection - ERS* - model) does not need depth knowledge of the application neither the environment; it is defined from the user point of view. In this regard, this study aims to guide the daily work of researchers to optimize the execution of their experiments in grid infrastructures.

Several *ERS* versions were implemented based on different *AI* algorithms. Two scenarios have been developed to evaluate the benefits of using the proposed model. Both scenarios are composed

DOI: 10.1002/cpe

by a test set in which different model parameters are varied. In every test, the proposed model is compared with the Traditional Resources Selection (*TRS*) in grid environments. From the obtained results it can be concluded that both a reduction of execution time and an improvement of the successfully completed task rate are achieved using our efficient strategy. This is a feasible and beneficial solution versus the *TRS*. Finally, from the first algorithm (*ERS-PA*) until the last one (*ERS-CA*) the model has evolved favourably and it has reached very satisfactory results.

Future work will avoid the use of P_α concept and will manage tasks directly, considering other grid services. Also new methodologies for improving the functionalities of the *ERS* modules will be considered.

ACKNOWLEDGMENT

María Botón-Fernández is supported by the PhD research grant of the Spanish Ministry of Economy and Competitiveness 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. D.B. Fogel *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd edition, IEEE Press, Piscataway, NJ, 2000.
2. I. Foster, C. Kesselman, *The Grid: Blueprint for a new Computing Infrastructure*, in: Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
3. 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), Euro-Par 2001, LNCS, vol. 2150, pp. 1-4. Springer-Verlag Berlin, Heidelberg, 2001.
4. I. Foster, *What is the Grid? A three Point Checklist*, in: GRIDtoday, Vol. 1, No. 6, pp. 22-25, 2002.
5. G. Wrzesinska, J. Maasen and H.E. Bal, *Self-adaptive applications on the grid*, in: 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Jose, California, USA, pp. 121-129, 2007.
6. D. Groen, S. Harftst and S.P. Zwart, *On the Origin of Grid Species: The Living Application*, in: Proceedings of the 9th International Conference on Computational Science: Part I, LNCS, VOL. 5544, Springer-Verlag Berlin, Heidelberg. pp. 205-212, 2009.
7. 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*, in: IEEE Transactions on Parallel and Distributed Systems, Volume 14, Issue 4, pp. 369-382, April 2003.
8. S.S. Vadhiyar and J.J. Dongarra, *Self adaptivity in Grid computing*, in: Concurrency and Computation: Practice and Experience, vol. 17, issue 2-4, John Wiley and Sons Ltd. Chichester, UK, pp. 235-257, 2005.
9. E. Huedo, R.S. Montero and I.M Llorente, *A framework for adaptive execution in grids*, in: Software-Practice & Experience, vol. 34, issue 7, John Wiley and Sons, Inc. New York, NY, USA, pp. 631-651, 2004.
10. 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*, in: DEXA'04 Proceedings of the Database and Expert Systems Applications, 15th International Workshop, IEEE Computer Society, Zaragoza, Spain, pp. 689-693, 2004.
11. Y. Gao, H. rong and J.Z. Huang, *Adaptive Grid Job Scheduling with Genetic Algorithms*, in: Future Generation Computer Systems, Vol. 21, Issue 1, Elsevier Science Publishers B. V. Amsterdam, The Netherlands, pp. 151-161, 2005.
12. D.M. Batista and L.S. da Fonseca, *a survey of self-adaptive grids*, in: IEEE Communications Magazine, vol. 48, issue 7, IEEE Press Piscataway, NJ, USA, pp. 94-100, 2010.
13. F. Xhafa and A. Abraham, *Metaheuristics for Scheduling in Distributed Computing Environments*, in: Metaheuristics for Scheduling in Distributed Computing Environments, Studies in Computational Intelligence, vol. 146, pp. 1-37, 2008.
14. F. Xhafa and A. Abraham, *Computational Models and Heuristic Methods for Grid Scheduling Problems*, in: Future Generation Computer Systems, vol. 26, issue 4, pp. 608-621, 2010.
15. J. Yu , R. Buyya , K. Ramamohanarao, *Workflow Scheduling Algorithms for Grid Computing*, in: Metaheuristics for Scheduling in Distributed Computing Environments, vol. 146, pp. 173-214, 2008.
16. J. Yu and R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, in: Journal of Grid Computing, vol. 3, issue 3-4, pp. 171-200, 2006.
17. M. Rahman, R. Ranjan, R. Buyya and B. Benatallah, *A Taxonomy and Survey on Autonomic Management of Applications in Grid Computing Environments*, in: Concurrency and Computation, Practice and Experience, vol. 23, issue 16, pp. 1990-2019, 2011.
18. Pegasus AI Based Scheduling: http://pegasus.isi.edu/
19. A-L. Barabási and A. Réka, *Emergence of Scaling in Random Networks*, in: Science, vol. 286, No. 5439, pp. 509-512, 1999.

DOI: 10.1002/cpe

20. S. Boccaleti, V. Latora, Y. Moreno, M. Chavez, D. Hwang, *Complex Networks: Structure and Dynamics*, : in: Physics Reports, vol. 424, n 4-5, Elsevier, pp. 175-308, 2006.
21. N. Mladenovic and P. Hansen, *Variable Neighbourhood Search*, in: Computers & Operations Research, vol. 24, Issue 11, Elsevier, pp. 1097-1100, 1997.
22. J.V. Neumann, *The Theory of Self-Reproducing Automata*, in: A.W. Burks (ed), University of Illinois Press Champaign, IL, USA, 1966.
23. M. Gardner, *Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life"*, in: Scientific American, Vol. 223, no. 4, pp. 120-123, 1970.
24. X. S. Yang and Y. Z. L. Yang. *Cellular automata networks*, in: Proceedings of Unconventional Computing, A. Adamatzky, L. Bull, B. De Lacy Costello, S. Stepney, C. Teuscher (eds.), Luniver Press, pp. 280-302, 2007.
25. The National Grid Initiative for Spain: http://www.es-ngi.es/
26. M. Botón-Fernández, F. Prieto Castrillo, M. A. Vega-Rodríguez, *Self-adaptive deployment of parametric sweep applications through a complex networks perspective*, in: ICCSA'11 Proceedings of the 2001 International Conference on Computational Science and Its Applications: Part II, LNCS, vol. 6783/2011, Springer-Verlag Berlin/Heidelberg, pp. 475-489, 2011.
27. A. Santiago and R.M. Benito, *An extended formalism for preferential attachment in heterogeneous complex networks*, in: Europhysics Letters, vol. 82, issue: 5, 2008.
28. M. Tomassini, *Generalized Automata Networks*, in: S. El Yacoubi, B. Chopard, and S. Bandini (eds.):ACRI 2006, LNCS, vol. 4173, Springer-Verlag Berlin, Heidelberg, pp. 14-28, 2006.
29. EGI. European Grid Infrastructure: http://www.egi.eu/
30. Ibergrid Project Web Page: http://www.ibergrid.eu/
31. W.H Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipies in C*, In: Press Syndicate of the University of Cambridge, New York, 1992.
32. gLite Middleware Web Page: http://glite.cern.ch/
33. EGEE Portal. Enabling grids for E-sciencE: http://www.eu-egee.org/