Noname manuscript No.
(will be inserted by the editor)

A Self-adaptive Resources Selection Model through a Small-World based Heuristic

María Botón-Fernández · Francisco Prieto-Castrillo · Miguel A. Vega-Rodríguez

Received: date / Accepted: date

Abstract The Small-world phenomenon is a principle in which seemingly distant nodes are linked by short chains of acquaintances. This property is found in a wide range of biological, social or natural networks. We proposed a self-adaptive model for solving the grid computing resources selection problem. A heuristic based on Small-World concepts is defined within this model. Grid computing infrastructures are distributed systems with heterogeneous and geographically distributed resources. The present approach selects the most efficient resources during the application execution for facing the environmental changes. The model is tested in a real European grid computing infrastructure. Finally, from the results that have been obtained during the evaluation phase it is possible to conclude that the model achieves a reduction in applications execution time as well as an increase in the successfully completed tasks rate.

Keywords Self-adaptivity · Grid Computing · Small-world Phenomenon · Optimization · Heuristic

1 Introduction

Grid computing environments [1] [2] are distributed systems consisting of heterogeneous resources used in a coordinated way and with a non-centralized control. This type of infrastructure connects resources with different geographical location by using different administrative domains (resources belong to

M. Botón-Fernández and F. Prieto-Castrillo
Ceta-Ciemat, Dept. Science and Technology.
Trujillo, Spain
E-mail: {maria.boton, francisco.prieto}@ciemat.es

M.A. Vega-Rodríguez
University of Extremadura, Dept. Technologies of Computers and Communications.
Cáceres, Spain
E-mail: mavega@unex.es

different centres). Each resource provider is responsible for managing its own elements (non-centralized control of resources). This fact leads to a dynamic and changing grid computing environment that modifies the availability and performance of its resources.

Grid computing applications face up a double heterogeneity within these systems. Firstly, there are different groups of resources according to their functionalities. Secondly, there are heterogeneous resources within a particular group due to the fact that they are provided by different centres (which also imply local administrative domains and local policies). Moreover, it should be taken into account the heterogeneity of jobs and the fact that grid computing applications compete for using different non-dedicated resources.

All these circumstances result in problems of resources discovery, resources selection, resources monitoring, etc. Then, applications require real-time information about grid computing infrastructures for dealing with the environmental changes. Notice that resources are added and removed in an unpredictable way and that also their software and hardware characteristics vary over time (both are environment changes resulting from local administration). For that reason, it has become a challenge to efficiently solve the resources management problems in grid computing environments owing to its dynamic nature and structure.

In recent years, the *adaptation* concept has been widely extended as an alternative for solving these problems. The grid computing community has proposed several adaptive approaches [14]-[24]. However, none of them has been globally extended across the different grid computing platforms.

The present contribution is focussed on solving the grid computing resources selection problem by choosing in an efficient way the resources that best fit the application requirements, without modifying the grid computing infrastructure. An Efficient Resources Selection (*ERS*) model is defined from the user point of view, by guiding applications during their deployment on grid computing infrastructures. That means, the model does not control or does not change resources behaviour.

During the evaluation phase two main objectives were fixed: on the one hand, achieving a reduction of the total execution time of applications. On the other hand, reaching an improvement of successfully finished tasks rate. Regarding these objectives, we are interested in identifying the most efficient resources in the shortest possible time. Furthermore, the selection process within our approach should be self-organizing and self-adaptive for facing grid computing infrastructures changes at runtime. That fact has motivated us to define a heuristic based on the Small-World (*SW*) phenomenon [3]-[5]. The available infrastructure resources are represented as nodes of a network (which is not a *SW* network) and they are linked when the model chooses them (i.e. a resource that finishes its execution is connected to the next resource to use) resulting in the corresponding set of edges. Then, it is expected that the model finds the most efficient resources considering the *six degrees of separation* idea. Specifically, the heuristic that has been implemented within the selection process applies the following SW concepts: the network contains hubs

(nodes with a high number of edges, that is to say, with a high degree) which are those resources used repeatedly by the model (the most efficient resources) and the edges are divided into *local* and *long-range* contacts, for applying the analogous search algorithms (Local Shortcuts Search Algorithm and Random Long-range Search Algorithm). The network starts with a set RT of z nodes (please see Section 3.2) and it grows during the application execution.

The present approach is denoted as *Efficient Resources Selection Model Based on Small-World* (*ERS-SW*). It must also be highlighted that the model is tested in a real European Grid Computing Infrastructure[1] belonging to a National Grid Initiative[2]. Previous versions of the model [12] [13] are also included and a comparative study, considering all *ERS* developed versions, is presented in the evaluation phase.

In conclusion, the main contributions of this work are:

- We propose a mathematical formulation for obtaining the resources efficiency in a grid infrastructure.
- This mathematical formulation combined with different heuristic techniques provides an efficient model for selecting the grid resources.
- This model is self-adaptive because it can adapt itself to the changing properties of a grid environment. The model achieves this self-adaptation without any control on the internal components of the grid infrastructure (that is to say, by using the user point of view).
- We propose a new version of this model by using a heuristic based on the Small-World phenomenon (*ERS-SW*). We also presented three previous versions of the model: *ERS-PA*, *ERS-VNS*, and *ERS-CA*.
- The novelty of *ERS-SW* compared with previous versions are: both the historical values of compute time μ_i and the historical value of successfully finished tasks ϵ_i are considered for measuring the fitness of a resource (see Eq. 1). Resources are evaluated in an independent way, not as a set. When calculating the workload of a resource it is taken into account not only the load produced by our experiments but also the local load generated by other applications which are running in the infrastructure.
- All versions of our model have been verified over real grid environments (not simulations) and compared with the standard scheduling technique of European grid computing infrastructures (denoted as *TRS* - Traditional Resource Selection), obtaining very interesting results: our model executes the tasks in less time and increases the number of successfully completed tasks.
- Furthermore, this model can also be used as an information system of infrastructure efficiency by using the statistics obtained as infrastructure status reports.

The rest of the paper is structured as follows. Section 2 summarizes related work using *adaptation*. Section 3 describes the *ERS* model, its formulation and self-adaptive selection. In Section 4 previous versions of our approach are

[1] http://www.egi.eu/about/ngis/
[2] http://www.es-ngi.es/

briefly discussed. Section 5 includes the evaluation of the model in a real grid computing infrastructure. Finally, Section 6 concludes the paper and indicates future work.

2 Related Work

As stated in the previous Section, there are several approaches which apply the *adaptation* concept for solving grid computing problems. Some of these works have developed adaptive frameworks to provide efficient jobs scheduling. There are other solutions which design autonomous systems, so that these systems would adapt to the dynamic infrastructure. Finally, other studies are focussed on applying *adaptation* within specific grid computing processes.

The work in [14] proposes an alternative for solving the selection process. It collects communication and processing times information periodically. Then, by using these metrics, two threshold (lower and upper) are calculated. The objective is to maintain the application efficiency between both thresholds. Resources are added or deleted based on this criterion.

In [15] an adaptive ability is provided by enhancing grid computing infrastructures. They present different options in order to avoid jobs restrictions: **1)** changing the infrastructure's design, **2)** developing a malleable jobs management and **3)** fostering the cooperation between users and infrastructure.

The study in [16] presents a methodology for managing grid computing applications autonomously. The concept of *living application* emerges from this work. The application makes decision on which tasks to do and which resources to use (it needs administrative privileges). These decisions are based on certain knowledge acquired by the application at runtime. Hence, the application operates in an autonomous way.

Grid computing systems are increasingly used for performing certain type of applications: multi-phase parallel applications. This fact has motivated the works in [17] [18] to develop efficient rescheduling frameworks. They proposed several strategies for deciding when and where rescheduling this type of applications.

In [19] it is described a version of the Advance Resource Connector (ARC) grid middleware which aims to solve several detected problems, such as system bottlenecks, by using *adaptation*.

A migration framework is exposed in [20] in which the resources load and application characteristics are considered. Novel policies are used for varying dynamically the resources load while maintaining the infrastructure performance.

Another intelligent framework is proposed in [21]. This one presents a new application model as well as it provides mechanisms for adapting dynamically during application execution. It is based on Gridway[3] project. The framework is also compared with similar approaches.

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

The study in [22] is focussed on enhancing the information service component denoted as *Monitoring and Discovery System* (*MDS3*) which is part of the *OGSA-based* (*Open Grid Services Architecture*) Globus Toolkit *GT3*. They present two self-adaptive notification algorithms to prevent the *Index Service* from overloading.

The AppLeS (*Application Level Scheduling*) project [23] aims to investigate adaptive scheduling within grid computing environments. This approach includes static and dynamic resources information, scheduling techniques and performance predictions for allowing applications to adapt themselves at runtime.

Finally, a survey which gathers the existing grid computing self-adaptive mechanisms is described in [24]. Also, some recommendations for reaching an autonomous management are suggested in that work.

All these studies are focussed on improving the infrastructure performance by using different techniques: scheduling, migration, control policies, or notification policies (they are defined from the system point of view). However, the present contribution propose an *ERS* model which aims to guide applications during their execution in grid computing systems. This approach is defined at the user level, which means, it does not apply scheduling technique or migration policies, and it does not change the infrastructure behaviour (please see Table 1). Then, our model selects the most efficient resources without modifying or controlling them (we only use the user command set).

Table 1 Main differences between the related work and the *ERS* model.

Solution	New Policies	Control Infrast.	Modify Infr. Behaviour.	Change IS
Adapt. App [14]	-	X	X	-
Adapt. RMS [15]	X	X	X	-
Living App [16]	-	X	X	-
TPC App [17]	X	X	X	-
MPC App [18]	X	X	X	-
ARC Syst. [19]	-	X	X	-
Adapt. Syst. [20]	-	X	X	-
Framework [21]	-	X	X	-
MDS3 [22]	X	X	X	X
AppLeS [23]	-	X	X	-
ERS Model	-	-	-	-

3 The Self-adaptive Resources Selection Model

As stated, the present approach is focussed on enhancing the grid computing selection process by providing a self-adaptive capability to applications. The *ERS-SW* strategy is based on a complex network algorithm for discovering and

selecting the most efficient resources. Next, the main grid computing concepts are exposed for a better understanding of the proposed model.

Grid computing environments are distributed systems composed of heterogeneous resources belonging to several centres, each of them with a different geographical location. These systems provide computational power and storage capacity for massive computing applications.

A Virtual Organization (*VO*) is a group of institutions with a common objective. The *VO* manages a grid computing infrastructure for accomplishing this objective. Then, users should register in a *VO* for using grid computing resources.

Finally, the main grid components of a typical grid computing infrastructure are summarized.

- User Interface (*UI*): The access point to a grid infrastructure for users.
- Computing Element (*CE*): The scheduler which manages the jobs' queue within a resources centre.
- Worker Node (*WN*): Node in which tasks are finally executed. They are managed by *CEs* within a resources centre.
- Storage Element (*SE*): Element with storage capacity.
- Resource Broker (*RB*): Meta-scheduler which deals with computing load balancing and manages the infrastructure storage.
- Information System (*IS*): The global information system that maintains resources status and information.

3.1 The *ERS* Mathematical Formulation

The *ERS* model is based on the mapping between two work spaces. On the one hand, a task space J which includes the n independent and parallel tasks of the current application. Tasks only differ in the values of input parameters (we focus on parametric sweep applications in which tasks have similar characteristics). On the other hand, an heterogeneous and dynamic resource space R which consists of the m available resources of the current grid computing infrastructure.

At the beginning of applications execution, the model creates a subset (in a random way) of J, denoted as T, and sends it into execution. For every task of T a resource $r_\alpha \in R$ is selected in a uniform random way (because at that moment there are not efficiency metrics). These resources compose the subset labelled as RT. The mapping between elements from both spaces allows the model to associate different tasks to the same resource (i.e. it is a *many-to-one* relationship). Although initially the model sends a subset of J into execution, from now on, tasks are handled independently. That is to say, when a task t_α ends its execution, a resource r_α will be efficiently selected for the next one. By sending a subset of J at the beginning of the execution we expect to promote the model for acquiring the efficiency metrics rapidly (i.e. encouraging a faster learning). Notice that every task $t_\alpha \in J$ has associated a *lifetime* lt, so that the model does not wait indefinitely for them. This *lifetime* value depends on

the application to be executed and it is fixed by the user in the configuration file. In particular, it is the sum of the average job queue time in the grid computing infrastructure (which has been obtained by experimentation) and the execution time of a job in a local machine (which is set by the user).

For calculating the efficiency of grid computing resources, a linear mathematical formulation is defined. Please note that the model works with normalized parameters (and also in the previous versions which are described in Section 4). In Eq. 1 it is defined the efficiency value F_i of a particular resource i. It is based on two main parameters: on the one hand, the historical value ϵ_i of finished tasks[4] and, on the other hand, the historical value μ_i of processing time used for finishing those tasks. Both parameters have two *relevance* weights, a and b respectively, for allowing users to specify the priorities within their experiments. The value of such parameters (a and b) is specified by users from the command line when they execute an application. It must be expressed as a percentage.

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

Parameter ϵ_i (Eq. 2) is defined as the ratio of the successfully finished tasks SFt_i and the total number of assigned tasks At_i for the ith resource.

$$\epsilon_i = SFt_i/At_i \ . \tag{2}$$

Regarding μ_i, for each task j the processing time consumed by resource i is measured. This processing time (Eq. 3) is based on both the communication time $Tcomm_{i,j}$ between resource i and other grid computing services and the computation time $Tcomp_{i,j}$ for task j.

$$T_{i,j} = Tcomm_{i,j} + Tcomp_{i,j} \ . \tag{3}$$

Next, all these values of $T_{i,j}$ are used to calculate the average of processing time $\bar{\chi_i}$ (Eq. 4) for the number of successfully finished tasks SFt_i at that moment.

$$\bar{\chi_i} = (\sum_{j=1}^{SFt_i} T_{i,j})/SFt_i \ . \tag{4}$$

Finally, the historical parameter μ_i is defined as follows (Eq.5), applying also the tasks *lifetime*. It must also be highlighted that all these parameters are normalized.

$$\mu_i = (lt - \bar{\chi_i})/lt \ . \tag{5}$$

[4] Within the model, every task whose grid status is *Done* or *Aborted* is considered a *finished* task. Also tasks whose *lifetime* ends before they had finished are considered as *finished* tasks.

3.2 Applying the Small-World based Heuristic

As stated, the heuristic defined in the proposed *ERS* model is based on several concepts of the *Small-World* phenomenon [4] [5] for selecting the most efficient resources during applications execution. By considering the *Small-World* in search processes a new optimal search algorithm is defined. It is known as Small-World Optimization Algorithm (*SWOA*) [25]. This algorithm includes both a Random Long-range process and a Local Shortcuts Search process.

In the previous Section the rules that govern spaces J and R have been described. This mathematical formulation is applied within the selection process for measuring the efficiency of the resources that have been used. Moreover, the efficiency value of resources is applied in the Local Shorcuts Search Algorithm for creating the neighbourhood of an evaluated resource (from this neighbourhood the next efficient resource is selected). We have also introduced the next components within the selection process:

- A workload threshold ϖ. This threshold is used to change from a local search to a global one (from local shortcuts to random long-range) during the selection process. It is assumed that a resource is overloaded when its workload[5] value exceeds ϖ[6].
- Evaluated resource set S_E. It is a set where all resources that have been selected by the model during the application execution are registered in. Resources are ordered from lower to higher efficiency values. This resource set is used within the Local Shortcuts Search. Notice that as a network is being generated during the application execution, the resources that compose this set represent the nodes linked in the network.
- Unevaluated resource set S_{UE}. This set is complementary to the previous one, so that, the resources that have not been selected (non-linked nodes) until now are included in it. It is used within the Random Long-range Search algorithm.

Local Shortcuts Search Algorithm

This algorithm handles the S_E set and applies the *neighbourhood* concept. In S_E we consider that the neighbourhood of a particular resource is composed by the two nearest resources, which are denoted as neighbours (Figure 1). As stated, resources within S_E are ordered from lower to higher efficiency values, so that, the neighbours of a particular resource r_α are those with an efficiency value close to it (resources v_1 and v_2). The neighbour with a higher efficiency value is on the right (v_1 in Figure 1) and the neighbour with a lower efficiency value is on the left (v_2).

Hence, when a task ends its execution, the model tries to select an efficient neighbour of the corresponding resource r_α for performing a new task. First,

[5] The workload value of a resource is calculated considering the local load (derived from other applications running in the infrastructure) and the load produced by our experiments.

[6] As the model uses normalized values, this threshold is fixed at 1. That means, a resource is considered as overloaded when its capacity is being used at 100%. Also the workload value of every resource is normalized to determine if it exceeds the threshold.

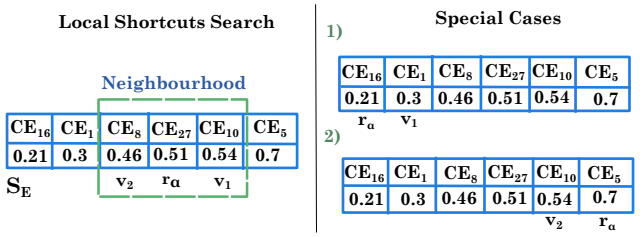

Fig. 1 Neighbourhood used within the Local Shortcuts Search Algorithm. The S_E set includes the resources that have been selected by the *ERS* Model.

the model applies the Local Shortcuts Search: the first option within that search algorithm is v_1 because it is more efficient than r_α. If v_1 is overloaded, then, the model selects v_2 as a new resource. If this resource is also overloaded the Local Shortcuts ends without result and the Random Long-range Search is applied. To determine if a resource is overloaded the ϖ threshold is applied. This threshold is updated whenever a resource is evaluated (due to the fact that its workload value is updated).

Random Long-range Search Algorithm

This other search algorithm is applied when the local search fails (when the neighbours of r_α are overloaded). In this case, the S_{UE} set is used to find a suitable resource for a new task. This resource is selected in a random way (because there are no efficiency metrics of these resources) and must fulfil a specific requirement: not exceed ϖ (not be overloaded). This algorithm is repeated until a resource with that characteristic is found.

There are two special cases within this search. On the one hand, during application execution all resources $\in R$ could be selected for performing tasks of space J; in this case S_{UE} is empty. On the other hand, resources from S_{UE} could have a workload value that exceed ϖ because of external applications. In both cases, the random search is performed over S_E.

Finally, it is specified the model execution flow as shown in Figure 2. After sending T into execution, tasks are monitored in an independent way. For that reason, when a task t_α finishes its execution the efficiency of the corresponding resource is calculated. Also the ϖ threshold is updated including this new efficiency value. Next, the model applies the heuristic based on SW and an efficient resource is selected for executing a new task. These processes are repeated until the whole space J is computed.

4 Previous Algorithms applied to ERS

In this section the previous versions of *ERS* are summarized. In each of them the selection process is based on different heuristic algorithms. Notice that in these versions the mathematical formulation for measuring the resources efficiency is very similar to *ERS-SW*. In this regard, the next three *ERS* ver-

PSEUDOCODE: ERS-SW ALGORITHM

Input: application tasks, infrastructure resources
Output: set of solutions

1. Determine spaces J and R;
2. Prepare set T;
3. Select randomly a set RT for executing T;
4. Launch T into execution;
5. while there are unprocessed tasks **do**
  5.1. Monitor tasks;
  5.2. If a task ends its execution **then**
    5.2.1. Update resource efficiency value;
    5.2.2. Update workload threshold;
    5.2.3. Apply SW heuristic algorithm;
      5.2.3.1 Apply Local Shortcuts Search Algorithm;
      5.2.3.2 If v_1 and v_2 are overloaded **then**
        5.2.3.2.1 Apply Random Long-range Search Algorithm;
    5.2.4 Assign new efficient resource to an unprocessed task;
    5.2.5 Launch a new task;
6. End while

Fig. 2 *ERS-SW* execution flow. Every time we measure resources efficiency and use them for selecting a new *CE* a self-adaptive capability is applied.

sions apply Eq. 6 for calculating the value of μ_i. The fitness of a resource i is calculated by using Eq. 1, the same as in *ERS-SW*.

$$\mu_i = (T_{max} - T_i)/(T_{max} - T_{min}) \ . \tag{6}$$

In these previous approaches, tasks and resources are handled by the model in different groups. That means, when all tasks belonging to T finish their executions, the model measures the efficiency of resources in RT. Therefore, T_{max} and T_{min} are the maximum and minimum T_i values achieved by resources $\in RT$. T_i is considered the processing time of a resource and it is calculated in a similar manner that the T_i of *ERS-SW*. Now, this time includes the computation times of every task assigned to that resource (At_i set, see Eq. 7).

$$T_i = \sum_{j \in At_i} (Tcomm_{i,j} + Tcomp_{i,j}) \ . \tag{7}$$

4.1 The ERS Model Based on Preferential Attachment

The *Preferential Attachment* algorithm (*PA*) [26] is used to generate random scale-free networks, which are presented in a wide range of natural and human-made systems. In this algorithm, a new node i is added to an existing node j

in the network with a probability proportional to the node's degree (i.e. the probability is proportional to the number of links that j already has).

Nodes with a higher degree value (which are denoted as *hubs*) tend to acquire more new links than those other with only a few links. For example, in a group of friends, the most popular acquires new friends more easily than the rest.

The *PA* is the first algorithm applied to the *ERS* model, resulting in the *ERS-PA* version [12]. In this case, the following assumptions were established for applying *PA* within the selection process:

- Resources belonging to space R are considered as nodes of a complex network built at runtime (during application execution).
- The degree of a node represents the number of times the corresponding resource has been selected for performing tasks during application execution.
- The probability of connecting a new node with an existing one depends on both the resource's degree and the resource efficiency value (see Eq.8).
- Hubs in this network are those resources with a higher efficiency value.

As stated, the efficiency value is used to link resources within our particular complex network. In this case, K_{max} is the maximum degree value for resources $\in RT$; k_i and F_i are the degree and fitness values of resource i

$$E_i = (k_i) \cdot F_i / K_{max} \ . \tag{8}$$

4.2 The ERS Model Based on Variable Neighbourhood Search

The metaheuristic known as *Variable Neighbourhood Search* (*VNS*) [27] is applied in global optimization problems. It is based on avoiding stagnation in a local search by changing the environment structure. The approach we present in this section is known as *ERS-VNS*.

In *VNS* a set of environment structures is denoted as N_k (which is a finite set with k_{max} environments) and a set of solutions for the k^{th} environment is named as $N_k(x)$. Now the main rules that govern the *ERS-VNS* heuristic algorithm are specified:

- Initialization:
  - Specify the set of environment structures that compose N_k.
  - Indicate an initial solution x.
  - Determine a stop condition: all T are processed?
  - Set $k = 1$.
- Repeat until all T are processed:
  - If $k = k_{max}$ then $k = 1$.
  - **Shaking:** generate a random solution x' from the k^{th} environment of x.
  - **Local search:** applying a local search technique using x' as initial solution. The resulting local optimum is denoted as x''.

- Use solution (resources) x'' for the tasks to execute.
- **Move or not:** if x'' is better than x do $x = x''$ and $k = 1$. Otherwise set $k = k + 1$.

As mentioned, in *ERS* we aim to find the most efficient resources for a particular application and, for that reason, we consider every *RT* as a local optimum solution. Moreover, we look for maximizing the fitness of the whole *RT* set (resources with efficiency values close to 1 are the most efficient ones). The fitness function for *RT* is defined as follows:

$$F(RT_\alpha) = (\sum_{i=1}^{d} F_i)/d \ . \tag{9}$$

It must also be highlighted that the fitness for every resource is obtained by applying Eq. 1. The parameter d specifies the number of resources within *RT* (the cardinality of *RT*). The next assumptions are also considered:

- Solution x is the initial *RT* set which has been selected for executing the first T set.
- Solution x' is obtained after performing a mutation process over x.
- Solution x'' is a new efficient set of resources (a new *RT*) for executing a new T set. By applying the local search process over x' this solution is acquired.

Furthermore, every environment structure handles other two parameters: p_k and q_k. On the one hand, parameter p_k indicates the difference or variation between solutions x and x'. This parameter varies in a non-decreasing way from an structure to another. On the other hand, q_k specifies the search range during the local search process. In the local search the available resources are ordered from higher to lower efficiency values. Next, the model selects the s most efficient resources to compose a candidate set. Parameter s is considered the q_k percentage of g (number of available resources). From this candidate set the new efficient resources to be part of the following *RT* are randomly chosen.

For determining the value of q_k the number of available resources g is considered. The value of p_k is set considering the cardinality of J and also g. Notice that both (p_k and q_k) are expressed as percentage and that it is possible to modify their values in the configuration file (by specifying their initial value and the increment to be applied). By using parameter q_k solution x'' is calculated.

Finally, in order to provide an advanced behaviour to *ERS-VNS* a threshold of fitness U is applied during the mutation process. This threshold determines the resources that should mutate, because they do not exceed this threshold (they are considered not efficient enough). Several experimentations have been performed in the corresponding grid computing infrastructure for obtaining the average efficiency value of resources. This value was stored in the U threshold.

4.3 The ERS Model Based on Cellular Automata

This third version is based on Cellular Automata methodology, in particular, the *ERS-CA* model [13] is based on John Conway *Game of Life* [28]. Furthermore, the grid computing infrastructure is managed as a cellular automata network [29] where *CEs* are represented by cells. In *ERS-CA* we improve the previous model versions (*ERS-PA* and *ERS-VNS*) by considering the resources workload. That means, when the model detects an overloaded resource, it identifies the corresponding site and tries to explore resources from a different one. A workload threshold wl is specified for detecting these situations (it is calculated by only considering the load generated by our experiments).

The proposed *CA* is composed by 8 subnetworks as shown in Figure 3, by considering geographical criteria. The number of subnetworks depends on the available resources (space R cardinality) within the infrastructure that has been used and it is specified by users in a configuration file. The number of components in each subnetwork also depends on such cardinality. It is established a minimal value of 3 elements in every one. Particularly, two cells which are neighbours in the *CA* are physical neighbour in the grid computing infrastructure.

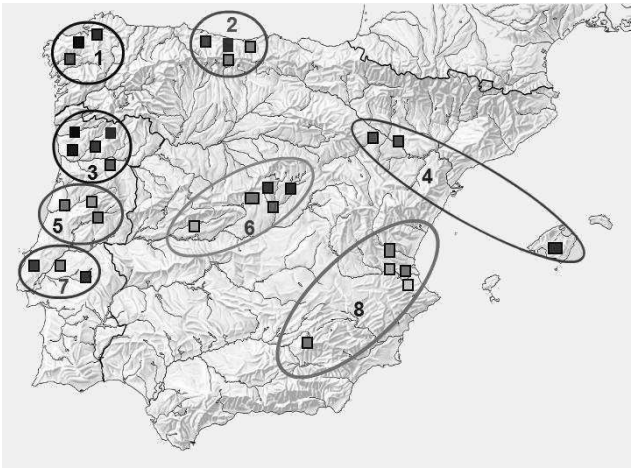

Fig. 3 Subnetworks that compose our *CA* considering the testbed infrastructure. Every subnetwork has at least 3 cells.

The cells in *ERS-CA* have three possible states: *alive, dead and inoperative*. An efficient resource is an *alive* cell while an inefficient resource is a *dead* cell. Resources which are overloaded or unavailable are *inoperative* cells. When the selection process is applied in each subnetwork, only two cells survive: those with the lowest workload values. Next, the rules that are applied in every subnetwork are specified:

– The pair of resources with minimum workload are considered living cells.

– Every cell that exceeds threshold *wl* dies.
– Any cell with its neighbourhood dead survives.

Finally, the living cells from all subnetworks compose a candidate set in which the following rules are performed for obtaining the new *RT*:

– Selective Pressure: the most efficient 50% of candidates promotes.
– Scout Resource: the model selects in a random way an unexplored resource (it has not been used until now).

5 Performance Evaluation

5.1 The Grid Computing European Infrastructure

As stated, the model has been evaluated in a real grid computing infrastructure which belongs to the European Grid Infrastructure (*EGI*)[7]. Particularly, the experiments have been performed in the Spanish National Grid Initiative (*ES-NGI*)[8]. We were affiliated to the generic Ibergrid *VO iber.vo.ibergrid.eu* which has about 30 *CEs*, a reasonable number of elements for evaluating the model. The Ibergrid infrastructure was officially created in 2007 and since then both countries (Spain and Portugal) have shared experience and expertise for providing grid computing services as a single organization along the Iberia area. Figure 4 represents the information about CPU and storage capacity of such infrastructure[9].

Site Classification					
CPU			STORAGE		
Type	Description	Sites	Type	Description	Sites
AAA	A minimum of 200 slots are available for the Ibergrid VOs (as a whole) at any time with HIGH PRIORITY. Jobs arriving to that site will enter in execution at the earliest possibility the cluster occupation permits	*ARASGRID, BIFI, BIFI-IBERGRID, CETA-GRID, IAA-CSIC, IFCA, IFIC, IFISC, NCG-INGRID-PT*	A	Persistent storage is available to the Ibergrid VOs (as a whole) up to a limit of "N TB" (with N > 1 TB).	*ARASGRID, BIFI, BIFI-IBERGRID, CETA-GRID, CESGA, CIEMAT-LCG2, CIEMAT-TIC, ESA-ESAC, IAA-CSIC, IFCA, IFIC, IFISC, UPV-GRYCAP*
AA	A minimum of 100 slots are available for the Ibergrid VOs (as a whole) at any time with HIGH PRIORITY. Jobs arriving to that site will enter in execution at the earliest possibility the cluster occupation permits	*CESGA, UPV-GRYCAP,NCG-INGRID-PT*			
A	A minimum of 50 slots are available for the Ibergrid VOs (as a whole).	*CIEMAT-TIC, ESA-ESAC, SGAI-CSIC*			
B	The VO is configured, but no resources are reserved. Only oportunistic runs are foreseen to happen.	*CAPFE-GRANADA, CIEMAT-LCG2, CFP-IST, IFAE, LCG-USC2, LIP-COIMBRA, LIP-LISBON, PIC, UAM-LCG2, UB-LCG2*	B	Non-persistent storage is available (as a whole), with a deadline of 30 days. Files will be deleted by the site administrator when older than 30 days. Each site willing to be in this category needs to configure at least 1TB on this status.	*CAPFE-GRANADA, CFP-IST, LCG-USC2, LIP-COIMBRA, LIP-LISBON, SGAI-CSIC, UAM-LCG2, UB-LCG2*
-	-	-	C	No support at the level of Storage Element is provided.	*IFAE, PIC*

Fig. 4 Classification of sites according to their computing and storage capacities.

The *ERS-SW* has been tested on a *UI* belonging to the Ibergrid infrastructure. This *UI* is a virtual machine provided by the Ceta-Ciemat [10] Center and

[7] http://www.egi.eu
[8] http://www.es-ngi.es
[9] All this information is collected from `http://ibergrid.lip.pt/`
[10] http://www.ceta-ciemat.es/

it is based on Scientific Linux 5.3 with python 2.4.3 (the *ERS* model has been implemented by using this programming language).

Within this phase we were interested in determining if the next two objectives were accomplished: a reduction in the application execution time and an increase in the number of successfully finished tasks. We consider both goals as important topics for grid computing users due to the fact that they improve the quality of the service.

Finally, the *ERS* model is compared with the standard selection technique used by European infrastructures, which uses the gLite[11] middleware. The process for selecting resources in this infrastructures is named as *match-making* and consists of choosing available CEs which are close to the corresponding input files (location criterion) and which satisfy the user requirements (qualification criterion).

5.2 Scenario 1

We designed this first scenario for determining the influence of size of T in the model learning. We introduced this set to speed up this learning process, so that, it is important to verify the accuracy of this assumption.

The scenario is composed of 5 tests where 10 real experiments are performed for every version (*ERS-SW*, *TRS*, *ERS-PA*, *ERS-VNS*, and *ERS-CA*). The graphical points in Figure 5 represent the average values of these sets of experiments. The size of J is fixed at 200 in all tests while the size of T is varied from a test to another (5, 10, 13, 20, 40). Finally parameters a and b were fixed at 60% and 40% respectively (we think users prefer executing successfully as many tasks as possible).

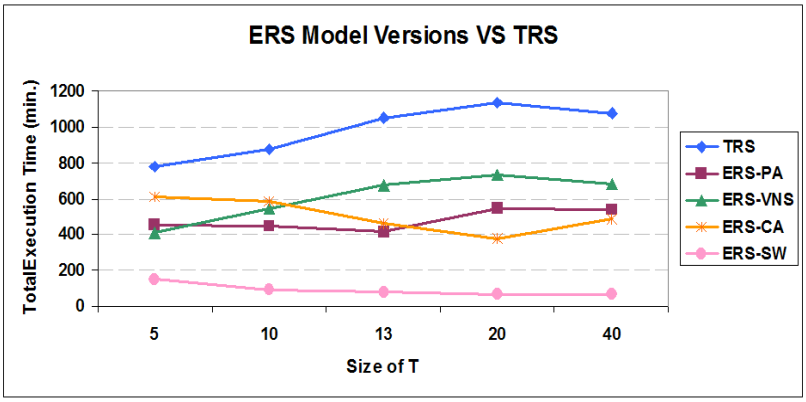

Fig. 5 Results that have been obtained in the first scenario. The *ERS-SW* totally improve the other versions.

[11] http://glite.cernch/

The results in Figure 5 show that all *ERS* versions have improved their execution times regarding *TRS*. *ERS-SW* reached the best execution time, with a significant average difference of 7 hours with respect to the other three *ERS* versions. The average difference with regard to *TRS* is about 15 hours. For high values of T size the *ERS-SW* classifies the efficient resources faster. All these *ERS-SW* results have been motivated by the idea of finding the most efficient resources in a short number of steps. Besides, in this version resources are selected based on both their efficiency value and their global workload value (by taking into account the load generated by other applications and the load produced by our experiments). Also, the independent evaluation of resources during the application execution leads to a fast knowledge of the infrastructure's state.

The *ERS-CA* version obtains the minimum values for high values of T size (13, 20, 40) because a larger amount of resources were evaluated at the same time (for a particular RT). Besides, as the load produced by our experiments has been taken into consideration the overloading of resources has been avoided. In contrast, *ERS-PA* performs better for values of low T size and it achieves the minimum value in size 13. In this version, the efficiency is based on nodes degree and fitness (the resources' load is not being considered). For values of high T size the model takes a long time to correct its decisions when a hub gets overloaded. For that reason the performance of the model is adversely affected. The *ERS-VNS* approach maintains a constant execution time difference with respect to *TRS*. It reaches its minimum values in the first two cases, in which there are more environment structure's changes due to the fact that more RT_α sets are evaluated (this fact implies a better approximation to the global optimum solution).

We can conclude that the size of T influences the performance of *ERS* versions involving a fast learning in several cases. The successfully finished tasks rate is also improved in every *ERS* version, as shown in Figure 6.

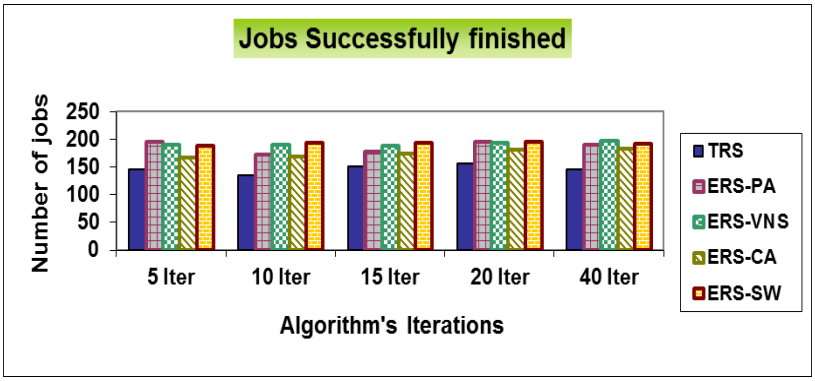

Fig. 6 Number of tasks finished in a successful way by the different *ERS* versions with respect to the *TRS* version.

In Table 2 several descriptive statistics concerning the total execution time for the different proposed versions of the model (ERS-SW, *ERS-PA*, *ERS-VNS*, and *ERS-CA*) and for the standard gLite selection (*TRS*) are included. It is possible to observe 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 computing infrastructures.

Table 2 Statistical values of *ERS-SW, ERS-PA, ERS-VNS* and *ERS-CA* in the corresponding tests. The mean is expressed in minutes.

TRS	Size 5	Size 10	Size 13	Size 20	Size 40
Mean	781	873	1053	1135	1080
Standard Deviation	252	178	350	133	14
Coefficient Variation	32%	20%	33%	12%	1%
ERS-SW	**Size 5**	**Size 10**	**Size 13**	**Size 20**	**Size 40**
Mean	151	92	77	63	65
Standard Deviation	24.61	14.91	7.56	7.7	18.3
Coefficient Variation	16%	16%	10%	12%	28%
ERS-PA	**Size 5**	**Size 10**	**Size 13**	**Size 20**	**Size 40**
Mean	451	449	413	546	540
Standard Deviation	48	63	55	42	48
Coefficient Variation	11%	14%	13%	8%	9%
ERS-VNS	**Size 5**	**Size 10**	**Size 13**	**Size 20**	**Size 40**
Mean	406	542	672	732	678
Standard Deviation	31.2	34.8	45	34.8	48.6
Coefficient Variation	8%	6%	7%	5%	7%
ERS-CA	**Size 5**	**Size 10**	**Size 13**	**Size 20**	**Size 40**
Mean	612	583	458	378	485
Standard Deviation	28.8	49.2	48	49.2	48
Coefficient Variation	5%	8%	10%	13%	10%

5.3 Scenario 2

The objective of this second scenario is to determine the range of grid computing applications in which the proposed model can report significant improvements. We expect to obtain suitable results for large production applications because this is the typical grid computing application scheme deployed in grid computing environments.

In this case, the scenario is formed by 6 tests in which the size of T is fixed at 10 because the *ERS* versions perform well at that point in the first scenario. The size of J varies from a test to another (50, 100, 200, 300, 400, 500). The values of parameters a and b are also fixed at 60% and 40% as in the first

scenario. As in the first scenario, 10 real experiments were performed for each version (*ERS-SW*, *ERS-PA*, *ERS-VNS*, *ERS-CA*, and *TRS*) in every test.

ERS-SW improves again the *TRS* execution time and also the execution times of the other *ERS* versions (Figure 7). Notice that this total execution time (in every model version) includes the application execution time and the time spent by the model for monitoring and classifying resources in an efficient way. In the last tests (300, 400 and 500) the results of *ERS-SW* are closer to the other versions. This is probably due to resources overload, because both *SW* search processes (Local Shortcuts and Random Long-range) are performed on S_E (probably because the two special cases specified in the Random-long Range Search - S_{UE} empty or overloaded - have occurred).

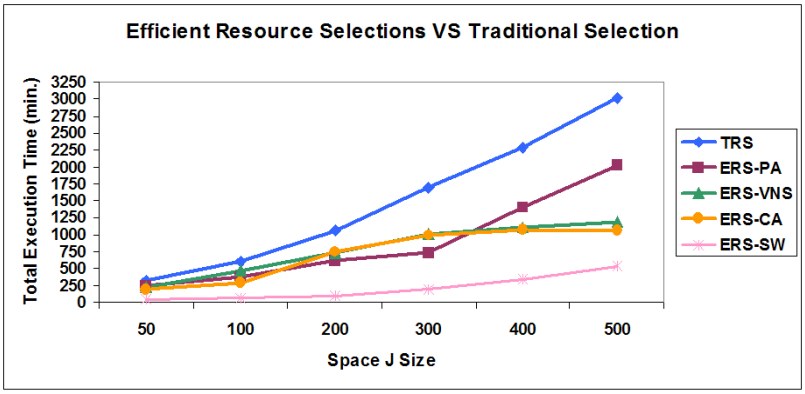

Fig. 7 Execution times values obtained within the second scenario.

ERS-PA, *ERS-VNS*, and *ERS-CA* have close values in points 50, 100, and 200. The execution times difference of these versions with regard to *TRS* grows as the size of J is increased. *ERS-VNS* and *ERS-CA* have a rather similar behaviour in all tests, achieving better results in the last ones. In the *ERS-VNS* case, when increasing the size of J more RT are evaluated so the global optimum fitness value can be reached. Concerning the *ERS-CA*, values of high J size lead to a better understanding of the infrastructure's state. The execution time difference between *ERS-PA* and *TRS* is almost constant from a size of 300 to 500 tasks. In these cases the same circumstances occur as in the previous scenario (hubs become overloaded and the *ERS-PA* takes time to appreciate it). Moreover, in this scenario the successfully finished tasks rate is also improved in all *ERS* versions with respect to *TRS* (see Figure 8). Also in this scenario the corresponding statistics are presented (see Table 3).

Finally, we can conclude that the two fixed objectives were accomplished and that the model is a reliable option for deploying applications in grid computing infrastructures. In this regard, applications with large tasks can take advantage of the proposed approach.

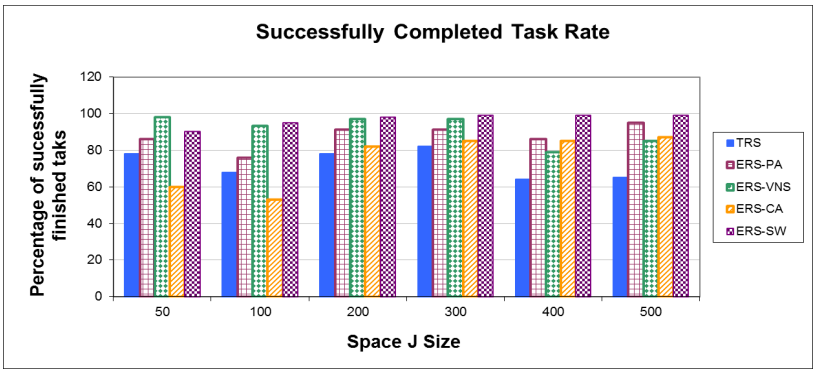

Fig. 8 Successfully finished tasks rate. *ERS-SS* achieves the best rate in most tests.

Table 3 Statistical values of the *ERS* versions and *TRS* within the second scenario.

TRS	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	328	611	1055	1695	2284	3019
Standard Deviation	59.4	120	178.8	271.8	661.8	699
Coefficient Variation	18%	20%	17%	16%	29%	23%
ERS-SW	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	41	60	92	202	335	545
Standard Deviation	11	12	15	41	18	41
Coefficient Variation	27%	20%	16%	20%	1%	8%
ERS-PA	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	241	379	615	731	1401	2022
Standard Deviation	42	64.8	45	49.8	43.2	48
Coefficient Variation	17%	17%	7%	7%	3%	2%
ERS-VNS	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	229	467	741	1008	1105	1189
Standard Deviation	37.8	43.8	28.8	19.2	48	48
Coefficient Variation	16%	9%	4%	2%	4%	4%
ERS-CA	Size 50	Size 100	Size 200	Size 300	Size 400	Size 500
Mean	190	294	752	999	1074	1060
Standard Deviation	34.2	46.8	39	28.2	49.2	48
Coefficient Variation	18%	16%	5%	3%	5%	5%

6 Conclusions and Future Work

The present contribution describes an Efficient and Self-adaptive Resource Selection Model for grid computing applications. The *Small-world* phenomenon is applied for enhancing the grid computing selection process. By using this model applications are able to adapt to the environmental changes. The proposed strategy selects the most efficient resources during application execution

without applying scheduling techniques (user point of view). The model does not control grid computing resources and it does not modify their behaviour. The article also includes a comparative study between the different *ERS* versions developed.

A real European grid computing infrastructure has been used as a testbed infrastructure for evaluating our model. Two scenarios were defined during this phase. In every test the *ERS-SW* is compared to the standard selection in European grid computing infrastructures *TRS*. From the obtained results it is possible to conclude that *ERS-SW* improves the infrastructure throughput, reducing the execution time and increasing the successfully finished tasks rate.

Future work will be focussed on improving the *ERS* model by applying other algorithms or heuristics within the selection process - making this process greedier - and by considering new grid computing services. As the *ERS* model is focussed only on parametric sweep applications, it would be interesting to test the model with other types of application that are executed in grid computing environments (like DAG - Directed Acyclic Graph - applications). Thus, the model would be general-purpose. Also future work will involve considering other circumstances during the experimental phase as task replication or analysing the *ERS* behaviour when taking into account load balancing.

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).

References

1. I. Foster, *What is the Grid? A Three Point Checklist*, GRIDtoday, Vol. 1, No. 6, pp. 22-25, 2002.
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. P. Erdos and A. Rény, *On the Evolution of Random Graphs*, in: Publications of the Mathematical Institute of the Hungarian Academy of Sciences, Vol. 5, pp. 17-61, 1960.
4. J. Kleinberg, *The Small World Phenomenon: an Algorithm Perspective*, in: Proceedings of The Thirty-second Annual ACM Symposium on Theory of Computing, pp. 163-170, Portland, Or, USA, 2000.
5. M. Newman, A-L. Barabási and D.J. Watts, *The Structure and Dynamics of Network*, in: Princeton University Press, 2006.
6. N. Mladenovic and P. Hansen. *Variable Neighbourhood Search*, Computers & Operations Research, Vol. 24, Issue 11, 1997, pp. 1097-1100.
7. P. Hansen and N. Mladenovic. *Variable Neighbourhood Search: Principles and applications*, European Journal of Operational Research Vol. 130, Issue 3, 2001, pp. 449-467.
8. P. Hansen and N. Mladenovic. *An introduction to Variable Neighbourhood Search*, in: S. Voss et al. (Eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer, 1999, pp. 433-458.
9. W.A. Beyer, P.H. Sellers and M.S. Waterman, *Stanislaw M. Ulams Contributions to Theoretical Theory*, Letters in Mathematical Physics, D. Reidel Publishing Company, Vol. 10, pp. 231-242, 1985.

10. A-L. Barabási and A. Réka, *Emergence of Scaling in Random Networks*, Science, AAAS, Vol. 286, No. 5439, pp. 509-512, 1999.
11. A. Santiago and R.M. Benito, *An Extended Formalism for Preferential Attachment in Heterogeneous Complex Networks*, Europhysics Letters, EPLA, Vol. 82, Issue 5, 2008.
12. 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.
13. 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.
14. 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.
15. 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
16. D. Groen, S. Harfst and S. Portegies Zwart, *On the Origin of Grid Species: The Living Application*, LNCS Vol. 5544, pp 205-212, 2009.
17. 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.
18. 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.
19. 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.
20. S.S. Vadhiyar and J.J. Dongarra, *Self Adaptivity in Grid Computing*, Concurrency and Computation: Practice & Experience, Vol. 17, Issue 2-4, John Wiley and Sons Ltd., Chichester, UK, pp. 235-257, 2005.
21. 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.
22. 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.
23. 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.
24. 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.
25. H. Du, X. Wu and J. Zhuang, *Small-World Optimization Algorithm for Function Optimization*, L. Jiao et al. (Eds.): ICNC 2006, Part II, LNCS 4222, pp. 264-273, 2006.
26. A-L. Barabási and A. Réka, *Emergence of Scaling in Random Networks*, Science, Vol. 286, N. 5439, pp. 509-512, 1999.
27. N. Mladenovic and P. Hansen, *Variable Neighbourhood Search*, Computer & Operations Research, Vol. 24, Issue 11, Elsevier, pp. 1097-1100, 1997.
28. M. Gardner, *Mathematical Games: The Fantastic Combinations of John Conway's New solitaire Game "Life"*, Scientific American, Vol. 223, No. 4, pp. 120-123, 1970.
29. X.S. Yang and Y.Z.L. Yang, *Cellular Automata Networks*, Proceedings of Unconventional Computing, A. Adamatzky, L. Bull, B. De Lacy Costello, S. Stepney, C. Teuscher (eds.), Luniver Press, pp. 280-302, 2007.