Performability-based Workflow Scheduling in Grids

Reza Entezari-Maleki[1], Kishor S. Trivedi[2], Leonel Sousa[3] and Ali Movaghar[4]

[1]*School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran*
[2]*Department of Electrical and Computer Engineering, Duke University, NC, USA*
[3]*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal*
[4]*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*
Email: entezari@ipm.ir, ktrivedi@duke.edu, las@inesc-id.pt, movaghar@sharif.edu

In this paper, the performance of a grid resource is modeled and evaluated using stochastic reward nets (SRNs), wherein the failure-repair behavior of its processors is taken into account. The proposed SRN is used to compute the blocking probability and service time of a resource for two different types of tasks, grid and local tasks. After modeling a grid resource and evaluating the performability measures, an algorithm is presented to find the probability mass function (pmf) of the service time of the grid resource for a program which is composed of grid tasks. The proposed algorithm exploits the universal generating function to find the pmf of service time of a single grid resource for a given program. Therefore, it can be used to compute the pmf of the service time of entire grid environment for a workflow with several dependent programs. Each possible scheduling of programs on grid resources may result in different service times and successful execution probabilities. Due to this fact, a genetic-based scheduling algorithm is proposed to appropriately dispatch programs of a workflow application to the resources distributed within a grid computing environment. Numerical results obtained by applying the proposed SRN model, the algorithm to find the pmf of grid service time, and the genetic-based scheduling algorithm to a comprehensive case study demonstrate the applicability of the proposed approach to real systems.

Keywords: Performability; Grid Computing; Stochastic Reward Net; Universal Generating Function; Task Scheduling

Received 5 November 2016; revised 24 August 2017; accepted 29 November 2017

1. INTRODUCTION

Grid computing integrates many resources distributed within multiple virtual organizations and administrative domains [1, 2]. Grid enables coordinated resource sharing and problem solving in dynamic and multi-institutional organizations. As well-known examples of grid computing projects, we can name Search for Extra-Terrestrial Intelligence (SETI@home) [3], and Large Hadron Collider (LHC) at European Organization for Nuclear Research (CERN) [4]. SETI developed a grid computing middleware where the program can be executed over multiple computers. The infrastructure was designed in such a way that a layman using the Internet could donate the unused computing power to the project. The middleware is known as BOINC (Berkeley Open Infrastructure for Network Computing) and was distributed under a GNU public license. The LHC Computing Grid is unlike the SETI@home project, it does not work with donated processing power. The computing grid is a closed one, since only certain organizations have access to it. To use the tremendous capabilities of the grid computing environment, grid users should deliver their own tasks to the grid resources directly or indirectly via a grid manager. In both cases, a scheduling algorithm is required to appropriately dispatch the tasks to the distributed resources. Since grid resources are very heterogeneous, and have different processing capabilities, task scheduling becomes quite important in grids [1, 2, 5]. Many static [6, 7, 8, 9, 10] and dynamic [11, 12, 13, 14]

scheduling algorithms have been proposed to schedule grid tasks among grid resources to achieve the required amount of Quality of Service (QoS). Some of the most important QoS parameters in grid environments are: availability of the resources, service reliability, security of a grid service, throughput of grid systems, performance of the environment in terms of overall completion time of tasks, waiting and service times of tasks, and energy [6, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24].

In order to deliver the tasks to a grid, distributed resources should be available to interact with grid users. Processors existing inside a resource can fail to execute grid tasks at any time. Therefore, the availability of a grid resource to serve grid tasks can be highly influenced by processor failures [2, 7, 16, 21, 22, 23, 24, 25, 26, 27]. On the other hand, grid resources should service local tasks submitted to them directly by local users in their administrative domains. When a user existing in a resource's administrative domain submits a task to the resource, it is served inside that resource along with the grid tasks submitted by grid users [1, 11, 19, 24]. Based on the processor management policies associated with a resource, different scheduling schemes can be considered to simultaneously dispatch grid and local tasks among processors of the resource. In some cases, a higher execution priority is assigned to serve local tasks over the grid tasks, but any kind of scheduling can be applied.

In addition to the availability of a grid resource, performance of the resource is one of the most important factors that should be taken into consideration, while developing a task scheduling algorithm. Generally, performance is the key issue for any system that serves user requests and can be considered according to the specification of the system and user expectations. In grid computing environment, performance evaluation mainly focuses on the time required to serve the tasks submitted by grid and local users. In this respect, several related measures such as mean service time and mean waiting time of the grid tasks can be considered. Moreover, blocking probability of grid tasks upon arrival is one of the interesting measures in such distributed computing systems in which designing a system with low blocking probability is important [6, 7, 18, 19, 26]. In traditional performance evaluation, each of these measures is assessed without any consideration of availability/reliability of a resource. Nevertheless, in highly distributed computing systems consisting of many independent resources (e.g. grid computing environments) each of the resources can fail while executing a task. Therefore, analyzing the pure performance of a grid resource tends to be optimistic, since it ignores the failure-repair behavior of the processors. On the other hand, pure availability analysis tends to be conservative, since performance is not taken into account. As a result, combined performance and availability, called performability [28, 29], evaluation of the grid environment can present a more realistic view of the system, and help to appropriately compute the blocking probability and the mean service time of the system for grid tasks.

In order to evaluate the performability of a grid resource, Stochastic Reward Nets (SRNs) are used in this paper. SRN is an extension of Stochastic Petri Nets (SPNs), which has the advantage of specifying and evaluating a real system in a compact and intuitive way. SRN has emerged as a powerful modeling paradigm in performance, availability and reliability analysis of fault tolerant computing and communication systems [27, 30, 31, 32]. The proposed SRN models a single resource in a desktop grid, considering both grid and local queues, and evaluates the performance of the resource, while the failure-repair behavior of its processors is taken into account. Since the proposed analytic model concretely depends on the structure of the resource, it models a resource of a desktop grid. In order to model other grid systems, for example grids of supercomputers, components of the model have to be changed to capture the requirements of the new system. Therefore, the proposed SRN has to be adapted to support grids of supercomputers. The performance measures computed for a single grid resource are the mean service time of the resource for grid tasks and the blocking probability of grid tasks upon arrival. Moreover, the probability of unsuccessful execution of grid tasks, which occurs due to processor failures in a resource, is also evaluated using the proposed SRN model. After modeling a grid resource using the proposed SRN and computing some useful measures, the universal generating function (u-function) [33] is used to represent the probability mass function (pmf) of service time of the resource for grid tasks. Using the proposed u-function, we can represent the pmf of service time of a grid resource for a single program that contains a set of grid tasks. Furthermore, the successful execution probability of a given program on a resource can be represented using this method.

Since the main objective of this paper is to propose a scheduling algorithm considering the performability of a grid resource, we use the u-function to schedule programs of a workflow application into the grid resources. The application considered in this paper is a general form of workflow applications in which the programs may have data or control dependencies on each other. Each program in the workflow application consists of a batch of tasks that are submitted to a grid resource after assigning the program to the resource. Knowing the computational complexity of a program (size of the program based on its tasks), and having the u-function of a grid resource, which represents the pmf of the service time of the resource for grid tasks, we can compute the service time and the successful execution probability of the program on the resource. Therefore, for each program/resource pair, we can define a u-function expressing the pmf of the service time of the specified resource for the

given program. Using this u-function, we propose a scheduling algorithm to assign the programs of a workflow application to the grid resources with the aim of minimizing the overall service time or maximizing the successful execution probability of all programs of the workflow application. Since the scheduling problem defined above is an NP-Complete problem [8, 20, 22, 23, 34], a Genetic Algorithm (GA) is proposed in this paper to explore the large search space of the problem in order to find a good enough solution.

The novelty of the paper is to utilize three aforementioned methods (SRN, u-function, and GA) in the context of scheduling problem in grid computing environments. The proposed approach proceeds in three steps. In the first step, we formally model a single grid resource with its internal structure, and then compute some performance measures of the resource while the availability of the processors is taken into account. Details considered in the model, such as arrival of both local and grid tasks, number of the processors, failure/repair behavior of the processors, and limited buffer for tasks, make the model more accurate. The proposed SRN can appropriately model and compute different performability measures for a single resource. Furthermore, our proposed model can easily handle different scheduling schemes for a single resource by trivial modification of the model components. The results obtained from the steady-state analysis of the proposed SRN model are used in the body of a u-function to find the pmf of service time of a single grid resource for grid tasks in the second step. Therefore, the mean service time and the successful execution probability of a program on a resource can be evaluated and represented in a mathematical way. In the third step, we bring single resources together to capture an entire grid environment, and use their related u-functions to compute the overall u-function of pmf of service time of the entire grid for a workflow application. The workflow studied in this paper contains dependent programs which should be dispatched among independent grid resources. The GA presented to schedule the programs to the grid resources uses the u-functions of the resources to find the overall u-function of pmf of service time of the grid to the workflow application for a possible scheduling of programs on resources. A simple example is presented to illustrate how the proposed methods can be applied to a grid environment. Afterwards, a comprehensive case study with large number of programs and resources is presented and solved with the proposed approach to show its applicability in real grid systems.

The rest of the paper is organized as follows. Section 2 introduces the related work on performance and dependability evaluation of distributed systems, as well as the state-of-the-art on task scheduling algorithms for grid systems. Section 3 provides a short introduction to SRNs. In Section 4, the proposed SRN model for a grid resource is provided, and the performability measures are discussed. Section 5 proposes an algorithm to find the pmf of service time of a grid resource for grid tasks and the GA-based scheduling algorithm. An illustrative example to show how the proposed SRN model and algorithms operate is provided in Section 6. Afterwards, a comprehensive case study with a wide variety of programs and resources and more realistic settings is presented in Section 7. Finally, Section 8 concludes the paper and presents future work.

2. RELATED WORK

This section reviews the related work done in the field of performance and dependability analysis of grid and cloud systems [7, 21, 23, 24, 25, 27, 30, 31, 32]. Moreover, the algorithms proposed to exploit analytic models for scheduling tasks on grid environments [6, 9, 10, 14, 16, 18, 19] are introduced.

Parsa et al. [18, 19] have proposed queuing network and Generalized Stochastic Petri Net (GSPN) solutions to model and evaluate performance of grid computing environments. The performance measures evaluated in [18] and [19] are the mean number of grid tasks waiting to receive service from the gird, and the total makespan of the environment, respectively. Both queuing network and GSPN models proposed in [18] only consider grid tasks submitted by grid users without paying any attention to the local tasks of the system, whereas the models proposed in [19] consider both types of tasks, grid and local tasks. The main drawback of the models proposed in [18] and [19] is that they only compute pure performance measures of the grid, and do not handle the possibility of one or more of the resources fail. Entezari-Maleki et al. [27] have proposed three different SRNs to model and evaluate the combined performance and dependability measure of a grid resource. The models of each grid resource were put together to capture the general model of a grid environment. Moreover, approximation techniques were used in [27] to overcome the state space explosion problem raised in the general model of the environment. Bruneo [30] has proposed an SRN model to investigate the performance of data centers in Infrastructure-as-a-Service (IaaS) cloud computing systems. The SRN proposed in [30] includes two major interesting functions of mathematical models for large scale systems, scalability and flexibility. Entezari-Maleki et al. [35] have proposed an analytic model based on Stochastic Activity Networks (SANs) to compute the performance and the power consumption of an IaaS cloud. The SAN model proposed in [35] models the Dynamic Voltage and Frequency Scaling (DVFS) technique when it allocates physical machines to the virtual machine requests.

Ghosh et al. [31] have proposed interacting stochastic sub-models to evaluate the performance of large scale IaaS clouds while workload of the cloud, system

characteristics and management policies are taken into consideration. In the same context, another model has been proposed by Bruneo et al. [32] to compare two different energy policies in green clouds using SRNs. The SRN model presented in [32] considers physical machines which dynamically changing the state through three different power consumption pools, sleep, idle and running, and allocates virtual machines on top of those physical machines to the user requests. Levitin et al. [7, 25] and Dai et al. [26] have analyzed the performance and reliability of grid services using the u-function technique. In [7], a grid with star topology was considered with all resources directly connected to the grid manager, and a u-function representing the pmf of grid service time was proposed. The failure events of a resource and the communication link with the grid manager were considered in the model. In [26], the model presented in [7] was extended to handle a computational grid with tree structure. In the model presented in [26], common-cause failures in communication links were taken into account. Both models proposed in [7] and [26] consider only tasks with independent subtasks, thus do not support subtasks with data or control dependencies on each other. To solve this problem, precedence constraints on subtask execution were studied in [25].

Azgomi et al. [16] have presented a Colored Petri Net (CPN) to model the workflow of tasks' execution in grid computing, and compute the reliability of a grid service. Although the CPN proposed in [16] precisely investigates failure events of grid resources and its effect on the performance, it does not consider local tasks and only solves the problem for a case that a single resource is individually considered. Entezari-Maleki et al. [6] have proposed a Markov chain model to compute the makespan on grids, and mean response time of a single task. The main advantage of the model proposed in [6] is that it allows to consider more than one manager for the grid. Distributed grid managers can be considered as heads of the clusters shaping the entire grid. However, the model presented in [6] still ignores the failure-repair behavior of grid resources, and the performance is purely evaluated without taking dependability issues into account. Tao et al. [10] have proposed a dependable grid workflow scheduling which uses Markov chains to predict availability of a resource. Based on the results obtained from the proposed Markov chain, a reliability-cost driven workflow scheduling algorithm was presented for grids. Garg et al. [9] have proposed an adaptive approach for scheduling workflows to dynamic grid resources based on a rescheduling method. The approach proposed in [9] involves initial static scheduling, resource monitoring and rescheduling with the aim of achieving the minimum execution time for a workflow by taking into account the availability of hosts and links.

Since the optimal scheduling of tasks on grid resources is an NP-Complete problem, evolutionary optimization techniques have been used to solve the problem [8, 21, 22, 23]. Levitin et al. [22] have proposed a technique for computing the execution time of tasks in a grid with a star topology while the failure behavior of grid resources and communication links are taken into account. Afterwards, a genetic algorithm was proposed to partition the tasks to execution blocks distributed across the grid resources. Dai et al. [23] have considered a grid with tree structure, and the problem of task partitioning and dispatching on it. In order to provide an optimal solution for task partitioning and distribution among the resources, an algorithm based on graph theory, Bayesian approach, and the evolutionary optimization technique was presented in [23]. The algorithm was designed to achieve the desired level of service performance. Entezari-Maleki et al. [8] have proposed a genetic algorithm to dispatch the independent tasks on distributed resources in a grid. The proposed algorithm in [8] considers the performance of the environment as a fitness function, and tries to minimize the total makespan of the system by appropriately distributing tasks to the resources.

Entezari-Maleki et al. [21] have exploited SANs to model and evaluate the performance of a grid resource, when the availability of the resource is taken into account. The model presented in [21] was used to assess the failure probability of grid tasks, and the throughput and mean response time of a resource for grid tasks. Afterwards, an algorithm was proposed to find the makespan of the grid when executing independent tasks under a specific scheduling. Finally, the Simulated Annealing (SA) meta-heuristic was used to find a good enough solution for scheduling of independent tasks on grid resources according to the proposed SAN, and the algorithm presented to find the makespan of the grid for a predefined scheduling.

Other related methods and algorithms in this research field can also be found in the literature. In general, each of the methods presented in this area has its own strengths and weaknesses. One of the problems with the previously presented methods in grids is that they usually do not consider the existence of both local and grid tasks, and the simultaneous execution of them in grid resources [19, 21, 24, 27]. A work that considers both types of tasks only computes pure performance of the grid environment without paying any attention to the failure-repair behavior of resources [19]. Moreover, previously presented methods mostly focus on measuring the mean value of output parameters, ignoring the distribution function of the output values [6, 16, 18, 19, 21, 23, 24, 27, 30, 31, 32, 35]. Proposing only a framework to evaluate the performance and/or dependability of grids, disregarding the scheduling problem of dependent programs on grid resources, is another difficulty found in the previous research papers [7, 23, 25, 26, 27, 31], which is addressed in this paper.

3. OVERVIEW OF STOCHASTIC REWARD NETS

Petri Net (PNs) is a graphical tool for the formal description of systems whose dynamics are characterized by concurrency, synchronization, mutual exclusion, and conflict, typical features of distributed systems [36, 37]. A Petri net can be defined as a 5-tuple:

$$PN = (P, T, F, W, M_0)$$

where
$P = \{p_1, p_2, ..., p_m\}$ is a finite set of *places*,
$T = \{t_1, t_2, ..., t_n\}$ is a finite set of *transitions*,
$F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*,
$W : F \to \{1, 2, 3, ...\}$ is a *weight function* and
$M_0 : P \to \{1, 2, 3, ...\}$ is the *initial marking*.

The graphical representation of a PN contains two disjoint sets of components, designated by places and transitions. The places and transitions are represented by circles and bars (or filled rectangles), respectively. The *marking* of a place corresponds to the number of tokens in the place. The marking of the PN is represented by a vector that specifies the marking of each place in the net. After specifying the marking of a PN, we can analyze the dynamic behavior of the net. A place is defined to be an input (output) place of a transition if there is an arc from the place (transition) to the transition (place). An integer ($d \geq 1$), called arc multiplicity, can be associated with each arc in the net. By default, if this number is not shown on the arc, it means that the arc multiplicity is 1. A transition is said to be *enabled* if each of its input places contains at least as many tokens as that input arc's multiplicity. An enabled transition is ready to $fire$. When a transition fires, a number of tokens equal to the multiplicity of the corresponding arc is removed from each of its input places. Moreover, each of its output places are deposited a number of tokens equal to the multiplicity of the corresponding output arc. Each firing generates a new marking of the net. Structural extensions to PNs include *inhibitor* arcs (denoted by an arc with a circle), which connect places to transitions [38]. A transition can be enabled only iff the number of tokens in its inhibitor place is less than the multiplicity of the inhibitor arc.

In the aforementioned definition, all transitions can fire as soon as they are enabled. The enabled transition can fire and there is no priority between the enabled transitions. This definition of PNs, which is known as basic PNs, is adequate for verifying the system's properties, e.g., liveness, boundedness, invariance, and so on. In order to allow a quantitative evaluation of the system's behavior, PNs have been extended in various ways to incorporate time notion [39]. The time-augmented PNs can be classified further depending upon whether the times mentioned are deterministic or stochastic. In the first case, the class of such PNs is called Timed Petri Nets(TPNs), and in the latter, they are called Stochastic Petri Nets (SPNs). More precisely, SPNs are PNs in which we always associate an exponentially distributed time delay to each transition [40]. In SPNs, all transitions are timed transitions. In order to overcome some quantitative problems, existing in SPN analysis and model immediate actions in some systems, Generalized Stochastic Petri Nets (GSPNs) have been introduced [38]. GSPNs have two different classes of transitions: *immediate* and *timed* transitions. Once enabled, immediate transitions fire in zero time. Timed transitions fire after a random, exponentially distributed enabling time, as in the case of SPNs. In the graphical representation of GSPNs, immediate transitions are drawn by bars or filled rectangles, while timed transitions are represented by white rectangular boxes. A marking of a GSPN is said to be *vanishing* if at least one immediate transition is enabled in that marking; otherwise, it is said to be *tangible*.

A Stochastic Reward Net (SRN) is obtained by associating *reward rates* with markings of a GSPN [41, 42]. SRNs allow the automated generation of Markov Reward Models (MRMs), facilitating the combined evaluation of performance and dependability of degradable fault-tolerant computer systems. We associate a reward rate ρ_i to every tangible marking of the SRN, then the expected reward rate at the steady-state can be computed as $\sum_i \rho_i \pi_i$, where π_i denotes the steady-state probability for the SRN to be in marking i. Several more extensions have been proposed for SRNs, namely for allowing multiplicity of arcs to be marking dependent and enabling functions or guards to be associated with transitions. The SRN models can be automatically transformed into MRMs, and then the steady-state and transient analysis of the obtained MRMs can produce the required measures of the original SRNs.

4. PERFORMABILITY EVALUATION OF A GRID RESOURCE

In order to compute the combined performance and availability of a grid resource, an SRN model is presented in this section. Afterward, the u-function technique is used to compute the probability mass function (pmf) of the service time of a grid resource for grid tasks.

A. SRN Model

The proposed SRN models a single grid resource with N processors inside the resource. Since there are two different types of tasks in each resource, grid and local tasks, we need a scheduling scheme to dispatch the tasks among the processors in a single resource. For this reason, the non-preemptive priority scheduling scheme is used in this paper to simultaneously schedule grid and local tasks to the processors of a resource. Any

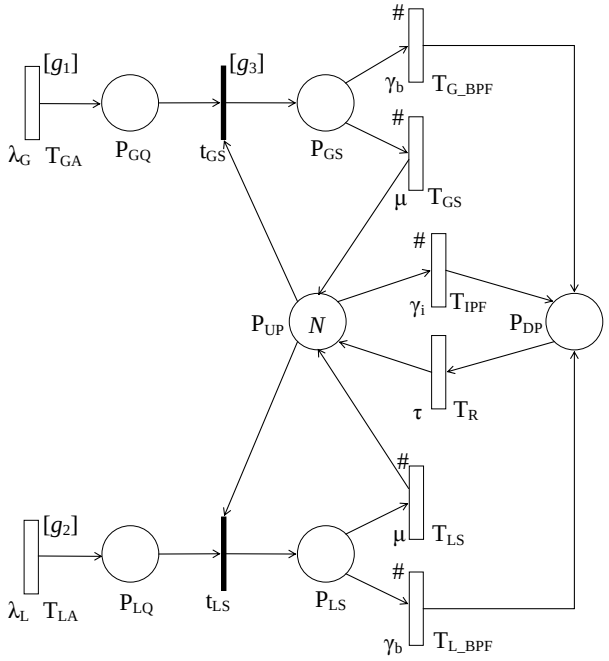

FIGURE 1. The proposed SRN model for a grid resource

scheduling scheme (e.g. random selection, preemptive priority and so forth) can be easily modeled by adding (removing) some components to (from) the proposed SRN. In non-preemptive priority scheme, a grid task is executed by one of the idle processors of the resource only if there is no local task in local queue of that resource. Fig. 1 shows the proposed SRN model, and Table 1 describes the places, transitions and arcs of the model. Input parameters of this model are: (1) the sizes of grid and local queues (M_G and M_L), (2) the arrival rates of grid and local tasks (λ_G and λ_L), (3) the number of the processors inside the resource (N), (4) the service rate of each processor (μ), (5) the failure rates of idle and busy processors represented by γ_i and γ_b, respectively with $\gamma_i < \gamma_b$, and (6) the repair rate of a failed processor (τ). It should be mentioned that the times assigned to all timed transitions follow exponential distribution.

The aim of the model is to evaluate the performance of a grid resource while its availability is taken into account. To achieve this, the failure-repair behavior of the processors in a grid resource should be considered in the model. Places P_{UP} and P_{DP} in Fig. 1 represent the *up* and *down* processors inside the resource, respectively. Since it is assumed to be N operational homogeneous processors inside a resource when the resource starts to serve the tasks, we put N tokens inside place P_{UP} to represent those processors. Transition T_{IPF} represents the failure event of idle processors. The pound sign (#) on the arc from place P_{UP} to transition T_{IPF} shows that the firing rate of this transition is marking dependent. So, the actual firing

TABLE 1. Elements of the SRN model presented in Fig. 1

Name	**Description**	**Rate/ Probability/ Initial no. of tokens**
P_{UP}	Operational processors	N
P_{DP}	Failed processors	0
P_{GQ}	Grid queue of the resource	0
P_{LQ}	Local queue of the resource	0
P_{GS}	Processors servicing grid tasks	0
P_{LS}	Processors servicing local tasks	0
T_{IPF}	Failure of idle processors	$[\#P_{UP}].\gamma_i$
T_R	Repair of failed processors	τ
T_{GA}	Arrival of grid tasks	λ_G
T_{LA}	Arrival of local tasks	λ_L
T_{GS}	Servicing grid tasks	$[\#P_{GS}].\mu$
T_{LS}	Servicing local tasks	$[\#P_{LS}].\mu$
T_{G_BPF}	Failure of processors servicing grid tasks	$[\#P_{GS}].\gamma_b$
T_{L_BPF}	Failure of processors servicing local tasks	$[\#P_{LS}].\gamma_b$
t_{GS}	Selecting grid tasks to be assigned to an idle processor	0.5
t_{LS}	Selecting local tasks to be assigned to an idle processor	0.5

rate of timed transition T_{IPF} is computed as $k.\gamma_i$, where k is the number of tokens in place P_{UP} and γ_i is the failure rate of an idle processor, as previously referred. Timed transition T_R represents the repair action of a failed processor. It is assumed that there exists only one repair facility for all processors inside a single resource.

As it can be seen in Fig. 1, there are two different lines of tasks arrivals. The first line is for grid tasks, submitted by grid users, and the second line is for local tasks, which are submitted by local users inside administrative domain of the resource. Transitions T_{GA} and T_{LA} show the arrival of grid and local tasks to the resource, respectively. There are two guard functions, g_1 and g_2, in the model associated with timed transitions T_{GA} and T_{LA}, respectively, to reflect the grid and local queue sizes of the resource. These functions are described in Table 2, where M_G and M_L denote the sizes of grid and local queues, respectively. Once transition T_{GA} (T_{LA}) fires, a token is deposited into place P_{GQ} (P_{LQ}) showing a grid (local) task has been

TABLE 2. Guard functions of the SRN model presented in Fig. 1

Guard Functions	Values
g_1	1 if $[\#P_{GQ}] < M_G$
	0 otherwise
g_2	1 if $[\#P_{LQ}] < M_L$
	0 otherwise
g_3	1 if $[\#P_{LQ}] = 0$
	0 otherwise

submitted to the resource and it is waiting to get service from the resource. Places P_{GQ} and P_{LQ} represent the grid and local queues of the resource, respectively. If there is a token in place P_{LQ} and there is at least one token in place P_{UP}, immediate transition t_{LS} fires; one token from P_{LQ} together with another token from P_{UP} are removed, and a token is put in place P_{LS}, representing a local task is getting service from the resource. Since higher execution priority is assigned to local tasks over grid ones, a grid task is assigned to an idle processor if there is no local task in local queue of the resource. The guard function g_3 described in Table 2 applies this condition to immediate transition t_{GS}. So, if there exists at least one token in both places P_{GQ} and P_{UP}, and there is no token in place P_{LQ}, transition t_{GS} fires; one token from P_{GQ} together with another token from P_{UP} are removed, and a token is deposited into place P_{GS}. Existence of a token in place P_{GS} (P_{LS}) shows that an idle processor has been allocated to a grid (local) task to serve the task.

Transitions T_{GS} and T_{LS} represent servicing of grid and local tasks inside the resource, respectively. The service rate of each individual processor is μ, which multiplied by the number of grid (local) tasks served in the resource gives the firing rate of the transition. This is shown by symbol # near transitions T_{GS} and T_{LS}. After firing the timed transition T_{GS} (T_{LS}), a token is removed from place P_{GS} (P_{LS}) and deposited into place P_{UP} to signal that one processor has already finished its job, and can be allocated to another waiting grid/local task. The failure events of busy processors are modeled by timed transitions T_{G_BPF} and T_{L_BPF}, for processors servicing grid and local tasks, respectively. The firing rates of these transitions are also marking dependent.

B. Performance Measures

The output of the SRN model presented in Fig. 1 is obtained by assigning appropriate reward rate to each feasible marking of it, assigning reward impulses to the transitions between the markings, and then computing the expected rewards in steady-state. Let ρ_i denote the reward rate assigned to marking i of the SRN model described above. If π_i represents the steady-state probability for the SRN model to be in marking i, then the expected steady-state reward rate can be computed as $\sum_i \rho_i \pi_i$. Moreover, if r_t denotes the reward impulse associated with transition t, then $\forall i \in \Omega$ (Ω is the state space of the underlying Markov chain of the SRN model shown in Fig. 1), $r_t(i)$ is the instantaneous reward gained when firing transition t in marking i [42, 43]. In most cases, a marking dependent reward impulse specification is not needed, and r can be considered as a real number. In this paper, the Stochastic Petri Net Package (SPNP) [44] is used to solve the numerical examples of the proposed SRN and obtain output measures. The interesting measures in the proposed model are the following.

Blocking probability of grid task arrivals. The steady-state blocking probability of grid task arrivals, P_b, can be computed by assigning the reward rate ρ_i to the proposed SRN model:

$$\rho_i = \begin{cases} 1, & [\#P_{GQ}] \geq M_G \\ 0, & otherwise \end{cases} \tag{1}$$

Failure probability of grid tasks. The steady-state failure probability of grid tasks, P_f, can be obtained by computing the actual firing probability of timed transition T_{G_BPF}. This value can be computed by associating impulse reward to firing timed transition T_{G_BPF}, which transfers underlying Markov chain of the SRN model shown in Fig. 1 from state i to state j. Assigning the impulse reward to *one* when T_{G_BPF} fires, and *zero* to all other timed transitions, we can use (2) to compute P_f.

$$r_{i,j} = \frac{\sum\limits_{t \in T : i \xrightarrow{t} j} r_t(i) \cdot \lambda_t(i)}{\sum\limits_{t \in T : i \xrightarrow{t}} \lambda_t(i)}, \tag{2}$$

where $\lambda_t(i)$ and $r_t(i)$ are the firing rate and the impulse reward associated with the firing timed transition t, receptively. The notation $\sum_{t \in T : i \xrightarrow{t}}$ represents all transitions accessing from state i by firing timed transition t [42]. Moreover, T denotes all timed transitions in the SRN model shown in Fig. 1, and $r_t(i)$ can be obtained as:

$$r_t(i) = \begin{cases} 1, & t = T_{G\_BPF} \\ 0, & otherwise \end{cases} \tag{3}$$

Throughput of a resource to grid tasks. Let $\pi(P_{GS} = k)$ denote the steady-state probability of there being k tokens in place P_{GS}. Hence, the actual throughput of a resource to grid tasks can be computed by (4).

$$Throughput = \sum_{k=1}^{N} \pi(P_{GS} = k) \times k \times \mu, \tag{4}$$

where N and μ denote the total number of processors inside the resource and the service rate of each

processor, respectively. There is a direct way in SPNP to compute the throughput of a transition. We use this capability of SPNP to compute the throughput of timed transition T_{GS}, which gives the throughput of the resource for grid tasks.

C. Universal Generating Function

The universal generating function, u-function, representing the probability mass function (pmf) of a discrete random variable Y is defined as a polynomial

$$u(z) = \sum_{k=1}^{K} \alpha_k z^{y_k}, \tag{5}$$

where the variable Y has K possible values and α_k is the probability that Y is equal to y_k [33].

In order to obtain the u-function representing the pmf of a function of two independent random variables $\varphi(Y_i, Y_j)$, composition operators are introduced. The composition operators determine the u-function for $\varphi(Y_i, Y_j)$ using simple algebraic operations on the individual u-functions of the variables [7, 16, 22]. The general form for all composition operators is shown in (6).

$$\begin{aligned} U(z) = u_i(z) \bigotimes_{\varphi} u_j(z) &= \sum_{k=1}^{K_i} \alpha_{ik} z^{y_{ik}} \bigotimes_{\varphi} \sum_{h=1}^{K_j} \alpha_{jh} z^{y_{jh}} \\ &= \sum_{k=1}^{K_i} \sum_{h=1}^{K_j} \alpha_{ik}\alpha_{jh} z^{\varphi(y_{ik}, y_{jh})} \end{aligned} \tag{6}$$

The u-function $U(z)$ represents all possible mutually exclusive combinations of realizations of the variables by relating the probabilities of each combination to the value of function $\varphi(Y_i, Y_j)$ for this combination. For example, for the function $add(Y_i, Y_j)$, the operator shown in (6) takes the form of

$$u_i(z) \bigotimes_{add} u_j(z) == \sum_{k=1}^{K_i} \sum_{h=1}^{K_j} \alpha_{ik}\alpha_{jh} z^{(y_{ik}+y_{jh})} \tag{7}$$

In this paper, the u-function $u(z)$ defines the pmf of the service time of a single resource for grid tasks. The service time of a resource for a grid task is defined as the time since the task is submitted to the resource until it leaves the resource. Therefore, the u-function representing the service time $\hat{t}$ of a grid resource for grid tasks takes the form:

$$u(z) = p(\hat{t}) z^{\hat{t}} + [1 - p(\hat{t})] z^{\infty}, \tag{8}$$

where $\hat{t}$ is the actual service time of the resource to grid tasks, and can be computed as $1/Throughput$ using (4), and $p(\hat{t})$ is the probability of the resource successfully serves the grid tasks. If the resource cannot serve grid tasks for any reason (e.g. the grid queue is full or a processor failure), the service time of the resource would be considered as infinite (∞). After modeling a grid resource using the proposed SRN to find the $Throughput$ measure using (4), we can compute probability $p(\hat{t})$ by

$$p(\hat{t}) = (1 - P_b) \cdot (1 - P_f), \tag{9}$$

where P_b and P_f are the blocking and failure probabilities of grid tasks, respectively, computed by applying (1) to (3) to the SRN model of a single grid resource. The reason behind (9) is that a grid task that is accepted by a grid resource with probability $1 - P_b$ will be executed by the resource with service rate $1/\hat{t}$ iff the servicing processors do not fail. Using (2) and (3), the probability that the processors servicing grid tasks do not fail is computed as $1 - P_f$. Therefore, the probability of successful execution of grid tasks in a resource can be computed by (9).

5. TASK SCHEDULING CONSIDERING RESOURCE PERFORMABILITY

This section presents a general case of workflow application to be scheduled on resources of a computational grid. With this purpose, an algorithm to find the pmf of the service time of a grid for a workflow application is presented. Afterward, a genetic-based scheduling algorithm is proposed to dispatch programs of a workflow to the grid resources, with the aim of minimizing the total service time or maximizing the probability of successful execution of the entire workflow application.

A. Workflow Representation

Directed Acyclic Graph (DAG) is a common representation of scientific workflows in computer science [1, 5]. The subclass of directed series-parallel graphs, *SP-graphs* for short, plays an important role in representing workflows [45, 46]. SP-graphs are restricted forms of DAGs which can be used to show the dependencies between programs of an application. SP-graphs are built by starting with single nodes. The single nodes may be combined in either series or parallel. The subgraphs obtained in this way may be combined in either series or parallel with other nodes or subgraphs [47]. Consequently, each SP-graph will have a unique source and a unique sink. It is worth to mention that a SP-graph is acyclic by definition. Our definition of series-parallel in this paper also implies that a SP-graph cannot have redundant edges. Hence, if there is an edge from node x to node y, there is no other path from x to y.

A workflow application can be represented by a SP-graph $G(V, E)$, where V is the program set $P = \{p_1, p_2, \ldots, p_n\}$, with n programs in the application, and E is the edge set representing the dependencies between the programs. The notation e_{ij} denotes an edge from the vertex p_i to p_j, and it

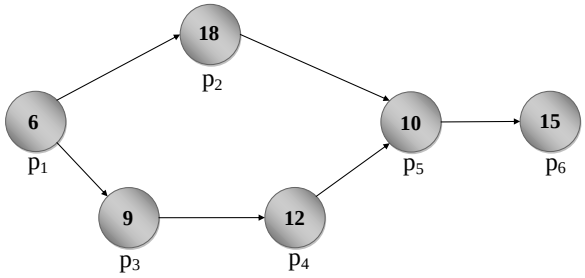

FIGURE 2. SP-graph representation of a sample workflow application

means that the program corresponding to the vertex p_j requires output data or results produced by the execution of program p_i. A SP-graph representation of a simple workflow application with *six* programs can be seen in Fig. 2. The number attached to program p_i represents the computational complexity (size) of p_i which is denoted by c_i. For example, the computational complexity of program p_1 is equal to 6 units, $c_1 = 6$. In fact, the unit assigned to each program is not important in our calculations, and it can be considered as any valid unit (e.g. *Million Instructions*), but we use the general word *task* to refer the computational complexity of a program.

In order to schedule programs existing in a workflow application to grid resources, a topological order of the programs should be found on a SP-graph. A topological ordering (or sorting) of a directed graph is a linear ordering of its vertices such that for every directed edge e_{ij} from vertex i to vertex j, i comes before j in the ordering. For instance, the orders $\{p_1, p_2, p_3, p_4, p_5, p_6\}$, $\{p_1, p_3, p_4, p_2, p_5, p_6\}$ and $\{p_1, p_3, p_2, p_4, p_5, p_6\}$ are all possible topological orders existing in the SP-graph shown in Fig. 2 which are valid sequences of program executions for this example. For our purpose, finding only one topological order suffices to be able to apply the proposed approach.

B. Algorithm to Find the pmf of Grid Service Time

Modeling grid resource R_j using the SRN proposed in Subsection A, the pmf of the service time of resource R_j can be computed as $u_j(z)$ by applying (8). Knowing the computational complexity of each program and the pmf of service time of each resource, the pmf of service time of a resource for a particular program is computed. Let T and S denote a possible topological order on a workflow application and a scheduling string, respectively. The scheduling string S, defined on a given topological order T, assigns each program of T to only one grid resource. Hence, the length of S is equal to the length of T, and each cell in S represents the number of the grid resource allocated to the relevant program. Suppose that the program p_i, with computational complexity c_i, is assigned to the resource R_j to be executed. Hence, the pmf of the service time of resource R_j for program p_i is computed as

$$u_{ij}(z) = p(\hat{t}_{ij}) z^{\hat{t}_{ij}} + [1 - p(\hat{t}_{ij})] z^{\infty}, \tag{10}$$

where $\hat{t}_{ij}$ is computed using (11).

$$\hat{t}_{ij} = c_i \cdot \frac{1}{Throughput}, \tag{11}$$

where $Throughput$ is obtained by applying (4) to the SRN of Fig. 1 for modeling the grid resource R_j.

After finding the pmf of the service time of an assignment, the pmf of total service time of a resource should be computed. We name this u-function as $U_j(z)$ for resource R_j. After the first allocation of resource R_j, $U_j(z)$ will be equal to the u-function resulted from the first allocation (e.g. $u_{ij}(z)$ if program i has been assigned to the resource R_j as a first assignment). Upon assigning other programs to a resource, the pmf of total service time of the resource should be updated by (7). For example, suppose program p_k is assigned to resource R_j as a second allocation of resource R_j in scheduling string S. Similar to (10), we can compute $u_{kj}(z)$ representing the pmf of the service time of resource R_j for program p_k when there is no load on resource R_j. Since the resource R_j has been allocated to execute program p_i before executing program p_k, the successful execution of program p_k depends on the successful execution of program p_i. Therefore, the pmf of total service time of resource R_j which is allocated to execute programs p_i and p_k is computed using (12).

$$\begin{aligned} U_j(z) = u_{ij}(z) \bigotimes_{add} u_{kj}(z) = p(\hat{t}_{ij}) p(\hat{t}_{kj}) z^{(\hat{t}_{ij} + \hat{t}_{kj})} \\ + [1 - p(\hat{t}_{ij}) p(\hat{t}_{kj})] z^{\infty} \end{aligned} \tag{12}$$

The procedure mentioned above is applied to all programs in the workflow application for computing the pmfs of total service time of all resources. Afterward, the *add* operator is applied to the u-functions of grid resources which have at least one assignment. Let $\Omega = \{R_1, R_2, \ldots, R_r\}$ denote the set of grid resources which have been allocated to execute at least one program. In other words, resource $R_j \in \Omega$ iff there exists a program p_i assigned to resource R_j in the scheduling string S on topological order T. Finally, the pmf of service time of the entire grid for a workflow application is computed by applying (13).

$$U(z) = U_1(z) \bigotimes_{add} U_2(z) \bigotimes_{add} \ldots \bigotimes_{add} U_r(z) \tag{13}$$

Algorithm 1 summarizes the above procedure.

C. Genetic-based Scheduling Algorithm

The u-function representing the pmf of the service time of entire grid for a given workflow application can be obtained by applying (13) to a specific scheduling string. Changing the scheduling string results in different u-functions, and consequently, different total

Algorithm 1 The procedure proposed to find the pmf of the service time of entire grid for a workflow application

1. Model all resources existing in the grid by the SRN model presented in Fig. 1.
2. Compute the measures P_b, P_f and *Throughput* of each resource.
3. Get workflow application W with all the information about its programs.
4. Find topological order T on workflow W.
5. Get scheduling string S.
6. For a given program p_i in T, find its corresponding resource R_j in S;
  6.1. Compute $u_{ij}(z)$ using (10)
  6.2. Update $U_j(z)$ using (12)
7. Find the set of grid resources which have at least one assignment (named Ω).
8. Compute the pmf of service time of the entire grid for workflow application W using (13).

service times and successful execution probabilities. Therefore, a scheduling algorithm should be developed to be executed by the manager to dispatch the programs to the resources. The main objective of this algorithm can be considered as finding a suitable scheduling string, with the aim of minimizing the total service time of the overall workflow or maximizing the successful execution probability of the workflow. In both cases, an NP-Complete allocation problem is defined. To solve this problem, the adoption of a Genetic Algorithm (GA) is proposed in this paper. GA is a promising heuristic and effective optimization approach to find near optimal solutions in large search spaces [8, 22, 48]. In order to propose a new GA, some predefined steps should be fixed.

In the first step, it is necessary to encode any possible solution of the problem as a set of strings named chromosomes. Each chromosome represents one possible solution for the problem. A set of chromosomes is referred to as a population. To represent the scheduling problem, an integer encoding is used in which each chromosome is represented by a vector of integers and each cell of the vector is considered as a gene. For a problem with n programs and m resources, the length of the vector is n and each gene can take a value between 1 and m, where the gene value represents the resource which is allocated to the program. For example, in a problem with six programs $(p_1, p_2, \ldots, p_6)$, as shown in Fig. 2, and three resources (R_1, R_2, R_3), a possible solution is 231221 which shows programs p_1, p_4 and p_5 are assigned to resource R_2, program p_2 is assigned to resource R_3, and programs p_3 and p_6 are assigned to resource R_1. It is worthwhile to mention that assigning all programs to a single resource is a valid assignment in this representation, but we avoid this case and only consider the assignments that use all available resources. In situations in which the number of programs (n) is much more than the number of resources (m), we can define a *low threshold* called l for a possible assignment to represent the minimum number of tasks assigned to each resource in a valid representation. For example, in string 231221, the low threshold factor is 1 ($l = 1$). If we consider $l = 2$, the previous string will be an invalid chromosome. One possible string for $l = 2$ is 132213 which assigns to each resource exactly two programs. In this case, there are $90 = \binom{6}{2} \times \binom{4}{2} \times \binom{2}{2}$ possible combinations for $n = 6$, $m = 3$, and $l = 2$. The value of the low threshold, l, can be chosen arbitrary, but assigning an appropriate value for l can help the GA to converge to the result faster, and may result more balanced load distribution among resources.

In the next step, an initial population of valid chromosomes is generated. The initial population (mostly a random set of chromosomes) is the first generation from which the evolution starts. The third step is to evaluate each of the chromosomes generated in the previous step. Each chromosome is associated with a fitness value, which in our algorithm is the total service time of a workflow or the probability of successful execution of the workflow represented by the chromosome. In other words, a fitness value is assigned to each of the chromosomes and the objective of the proposed GA is to find a chromosome with near-optimal fitness value. In our case, the fitness value can be computed by applying algorithm shown in Algorithm 1 to each possible chromosome. For example, if we consider the total service time of a workflow as the fitness value, chromosomes with smaller fitness values represent better solutions. Within this population, new solutions are obtained during the genetic cycle applying crossover and mutation operators. To select some of the chromosomes to breed a new generation, a selection method, such as roulette wheel selection, rank selection, tournament selection, should be used. Herein, the rank selection method is exploited, but any other selection methods can be used. In the crossover step, all chromosomes selected in the selection phase are paired up, and with a certain probability they are crossed over. With the crossover procedure used in the proposed algorithm, two crossover points are randomly selected, between 1 and n, for each couple of chromosomes. Code string from beginning of the chromosome to the first crossover point is copied from the first parent (assume $parent1$), the order of genes existing in the part from the first to the second crossover point are inherited from the second parent (assume $parent2$), and the rest is copied from the first parent ($parent1$). By applying this method, new generated offspring may inherit the best possible characteristics of the parents.

After crossover, each gene of a chromosome is mutated to any one of the codes, with a mutation probability. The mutation process transforms a valid chromosome into another valid chromosome that may or may not already be in the current population. The mutation method used in our algorithm just swaps

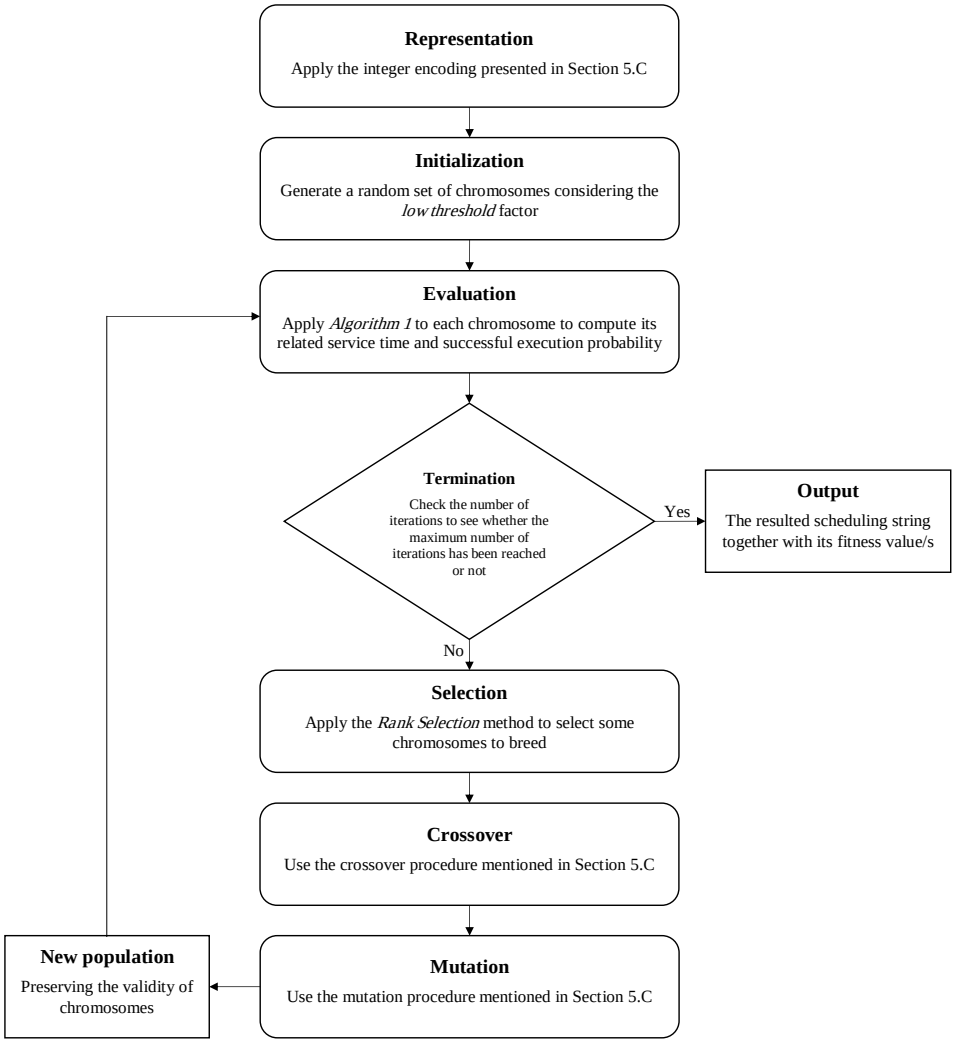

FIGURE 3. Steps of the proposed GA-based scheduling algorithm

genes initially located in two randomly chosen positions of a chromosome selected for mutation. After crossover and mutation, the new population is generated and the fitness function is applied to the new chromosomes. Crossover and mutation procedures should preserve the validity of newly obtained strings. Both crossover and mutation methods introduced above satisfy this requirement, so they can be used to generate a new population. The loop of chromosome generation is terminated according to a predefined criteria. The elite chromosome as a good enough solution together with its corresponding fitness value as an optimized measure is returned. Herein, the good enough solution is the scheduling string produced by the proposed algorithm, and the optimized measure is either the service time or the successful execution probability. We use the number of generations as a termination condition, but in general, other criteria such as allocation constraints (e.g., time), manual inspection, and combination of these conditions can be adopted. Fig. 3 graphically represents the steps of the proposed GA.

6. AN ILLUSTRATIVE EXAMPLE

A simple numerical example is presented in this section to illustrate the application of the proposed approach, and how it works. Suppose a grid computing environment with three different resources, R_1, R_2 and R_3, without any connection between the resources. The programs are dispatched by a manager that has a direct connection to each resource. The specifications of the resources are shown in Table 3. These values are just simple numbers, which can be replaced according to the features of real systems. As it can be seen in Table 3, the failure rate of a busy processor is higher than that of an idle processor for all resources. Also, the arrival rate of grid tasks is assumed to be greater than the arrival rate of local tasks in all resources, because it is reasonable to think that a resource joins a grid environment whenever its local load is not significant. All resources are modeled with SRN represented in Fig. 1, and the measures defined in Subsection B are computed for them. Table 4 shows the performance measures obtained from the steady-state analysis of

TABLE 3. Configuration of sample resources

Parameters	R_1	R_2	R_3
Grid tasks arrival rate (λ_G)	10.0	8.0	13.0
Local tasks arrival rate (λ_L)	6.0	6.0	7.0
Service rate of a processor (μ)	4.0	2.0	5.0
Grid queue size (M_G)	20	15	30
Local queue size (M_L)	20	20	15
Number of processors (N)	4	7	3
Failure rate of an idle processor (γ_i)	0.05	0.02	0.01
Failure rate of a busy processor (γ_b)	0.2	0.1	0.3
Repair rate of a processor (τ)	2.0	3.0	5.0

TABLE 4. Measures obtained from the steady-state analysis of SRNs modeling sample resources

Measures	Steady-State Values		
	R_1	R_2	R_3
Blocking probability of grid task arrivals (P_b)	0.14659	0.07858	0.39236
Failure probability of grid tasks (P_f)	0.01455	0.01253	0.014577
Throughput of the resource to grid tasks	8.12778	7.020380	7.45216

SRN models.

After modeling the grid resources using the proposed SRN, the pmf of the service time of each resource for grid tasks is computed by applying (8). The u-functions, representing the pmf of service time of resources R_1, R_2 and R_3 are shown in (14).

$$\begin{cases} u_1(z) = 0.84z^{\frac{1}{8.13}} + 0.16z^{\infty} \\ u_2(z) = 0.91z^{\frac{1}{7.02}} + 0.09z^{\infty} \\ u_3(z) = 0.60z^{\frac{1}{7.45}} + 0.40z^{\infty} \end{cases} \tag{14}$$

Suppose there is a workflow application with six different programs as shown in Fig. 2, which should be scheduled on resources R_1, R_2 and R_3. In order to schedule the programs of the workflow, the topological order $T = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ is considered. Moreover, a possible scheduling string named S_1 is assumed as one of the chromosomes of the initial population of the proposed GA. According to the chromosome representation rules of the proposed GA mentioned in Subsection C, a possible valid coding for S_1 is $S_1 = 121323$.

Considering the scheduling string S_1, in the first assignment, program p_1 is assigned to resource R_1. According to (10) and (11), the pmf of the service time of resource R_1 for program p_1 is computed as

$$u_{11}(z) = 0.84z^{0.74} + 0.16z^{\infty} \tag{15}$$

Based on the definition of the pmf of the total service time of a resource, presented in Subsection B, $U_1(z)$ can be written as

$$U_1(z) = 0.84z^{0.74} + 0.16z^{\infty}, \tag{16}$$

since it is the first allocation of resource R_1 to programs.

According to the scheduling string S_1, program p_2 is assigned to resource R_2, after assigning program p_1 to resource R_1. The pmf of the service time of resource R_2 for program p_2 is computed as:

$$u_{22}(z) = 0.91z^{2.56} + 0.09z^{\infty} \tag{17}$$

Similar to (16), we can write $U_2(z)$ as:

$$U_2(z) = 0.91z^{2.56} + 0.09z^{\infty} \tag{18}$$

The third assignment in S_1 is allocating resource R_1 to program p_3. Hence, the pmf of the service time of resource R_1 for program p_3, considering the situation in which there is no load on resource R_1, is computed as:

$$u_{31}(z) = 0.84z^{1.11} + 0.16z^{\infty} \tag{19}$$

According to (12), we can compute $U_1(z)$ using (15) and (19) as:

$$U_1(z) = u_{11}(z) \underset{add}{\otimes} u_{31}(z) = 0.71z^{1.85} + 0.29z^{\infty} \tag{20}$$

This procedure is continued till the last assignment ($p_6 \rightarrow R_3$). After assigning all programs, the pmf of the service time of all three resources are obtained. These u-functions are shown in (21).

$$\begin{cases} U_1(z) = 0.71z^{1.85} + 0.29z^{\infty} \\ U_2(z) = 0.83z^{3.99} + 0.17z^{\infty} \\ U_3(z) = 0.36z^{3.62} + 0.64z^{\infty} \end{cases} \tag{21}$$

Finally, after applying (13) to (21), the pmf of service time of the sample grid for the given workflow application is computed as

$$U(z) = 0.21z^{9.46} + 0.79z^{\infty}, \tag{22}$$

which shows that the hypothetical grid executes the workflow application shown in Fig. 2 in 9.46 time units (e.g. sec), with the probability of 0.21, according to scheduling S_1.

If we change the string S_1 to different scheduling, $U(z)$ would also be different. The resulting u-function strongly depends on both the factor l for setting the scheduling string and the fitness value considered for the proposed GA. As an example, if we set the low threshold factor to 2 ($l = 2$), as it was for S_1, and consider the service time of the workflow as the fitness value and try to minimize this value using the proposed

TABLE 5. The best and worst scheduling strings for $l = 0, 1, 2$, in the viewpoint of the grid service time

l	Scheduling String	The Resulting u-function
0	111111	$U_{best}(z) = 0.35z^{8.61} + 0.65z^{\infty}$
	222222	$U_{worst}(z) = 0.57z^{9.97} + 0.43z^{\infty}$
1	213111	$U_{best}(z) = 0.27z^{8.83} + 0.73z^{\infty}$
	123222	$U_{worst}(z) = 0.35z^{9.78} + 0.65z^{\infty}$
2	212331	$U_{best}(z) = 0.21z^{9.15} + 0.79z^{\infty}$
	121332	$U_{worst}(z) = 0.21z^{9.50} + 0.79z^{\infty}$

GA, we can reach the scheduling string $S_2 = 212331$. The u-function corresponding to the scheduling string S_2 is $U(z) = 0.21z^{9.15} + 0.79z^{\infty}$. The worst case for $l = 2$ in the perspective of service time is $S_3 = 121332$, which leads to $U(z) = 0.21z^{9.50} + 0.79z^{\infty}$. It turns out that the successful execution probability for all possible scheduling strings when $l = 2$ is 0.21. This probability changes when the parameter l is modified. Table 5 shows the best and worst service times for this example when $l = 0, 1, 2$, as well as the corresponding scheduling strings.

As it can be seen in Table 5, when the factor l is set to *zero*, the algorithm assigns all programs to the fastest resource, R_1, to reach the minimum service time for the workflow application. On the other hand, to reach the maximum probability of successful execution, the algorithm assigns all programs to the most reliable resource, R_2. In both cases, we have single point of failure because all programs have been assigned to a single resource. In order to avoid this situation and provide a more balanced schedule, the low threshold factor, l, should be set to an appropriate value. One proper value for l can be considered as $\lfloor \frac{n}{m} \rfloor$ where n and m are the numbers of programs and resources, respectively.

7. NUMERICAL RESULTS

In this section, a larger grid environment with a higher number of programs is studied, and then a comparison study with the state-of-the-art is performed.

A. A Comprehensive Example

Herein, a grid environment with 100 resources and a workflow application with 1000 dependent programs are considered as a more realistic case study. None of the resources are connected to the others, and they only can get programs from the manager. To generate the workflow application, the open-source software for visualizing and analyzing large networks graphs, *Gephi* [49] is used. Using this tool and setting the number of nodes to 1000 and the wiring probability of the graph to 0.2, a DAG application in which the nodes are connected to each other in series or parallel is generated. The size of each program of the DAG is randomly generated in the range $[1, 100]$. After generating a random workflow, the specification of internal configuration of each resource should be determined before being able to use the proposed SRN model to calculate the probabilities P_b and P_f, and the *Throughput* of the resource. The configurations of resources used are shown in Table 6. As can be seen in this table, for each parameter is specified a range, the exact value for that parameter is selected randomly within that range. Since none of the workloads and logs reported from real grid systems contains all our required detailed information of the resources and workflows, we have looked at some famous datasets in this area to set realistic values and ranges for the parameters of our case study. Five major workloads and logs from The Failure Trace Archive [50] (*deug* and *lri* logs), Parallel Workloads Archive [51] (*OSC Cluster* and *CEA CURIE* workloads) and The Grid Workloads Archive [52] (*NorduGrid* workloads) have been used. However, some of the parameters, such as grid and local queue sizes of a resource, which are basically related to the inner structure of the resource could not be found in these datasets. In these cases, we use random numbers to have a fair simulation.

TABLE 6. Configuration of resources in the comprehensive case study

Parameters	*Ranges*
Grid tasks arrival rate (λ_G)	$[5, 15]$
Local tasks arrival rate (λ_L)	$[1, 5]$
Service rate of a processor (μ)	$[5, 10]$
Grid queue size (M_G)	$[5, 20]$
Local queue size (M_L)	$[5, 10]$
Number of processors (N)	$[1, 4]$
Failure rate of an idle processor (γ_i)	$[0.005, 0.05]$
Failure rate of a busy processor (γ_b)	$[0.01, 0.02]$
Repair rate of a processor (τ)	$[5, 10]$

After modeling all resources using the proposed SRN model and computing the measures introduced in Subsection B, the u-function representing the pmf of the service time of a resource can be defined using (8). Having the workflow application and finding a topological order among its programs, the proposed GA is applied to find a suitable scheduling of programs of the application on grid resources. In this case study, the size of the initial population, and the crossover and mutation probabilities are set to 100, 0.9 and 0.2, respectively. As mentioned earlier, the number of the iterations is considered as the termination condition of the proposed GA. In this example, the maximum

TABLE 7. U-functions resulted from scheduling an application with 1000 programs on a grid with 100 resources

l	**Fitness value**	**The Resulting u-function**
1	Probability	$U(z) = 0.447z^{7699} + 0.553z^{\infty}$
	Time	$U(z) = 0.393z^{7030} + 0.607z^{\infty}$
2	Probability	$U(z) = 0.443z^{7676} + 0.557z^{\infty}$
	Time	$U(z) = 0.394z^{6956} + 0.606z^{\infty}$
3	Probability	$U(z) = 0.441z^{7687} + 0.559z^{\infty}$
	Time	$U(z) = 0.390z^{7016} + 0.610z^{\infty}$
4	Probability	$U(z) = 0.438z^{7620} + 0.562z^{\infty}$
	Time	$U(z) = 0.381z^{7037} + 0.619z^{\infty}$
5	Probability	$U(z) = 0.436z^{7599} + 0.564z^{\infty}$
	Time	$U(z) = 0.394z^{7078} + 0.606z^{\infty}$
6	Probability	$U(z) = 0.432z^{7606} + 0.568z^{\infty}$
	Time	$U(z) = 0.395z^{7100} + 0.605z^{\infty}$
7	Probability	$U(z) = 0.426z^{7552} + 0.574z^{\infty}$
	Time	$U(z) = 0.399z^{7110} + 0.601z^{\infty}$
8	Probability	$U(z) = 0.426z^{7475} + 0.574z^{\infty}$
	Time	$U(z) = 0.397z^{7161} + 0.603z^{\infty}$
9	Probability	$U(z) = 0.418z^{7427} + 0.582z^{\infty}$
	Time	$U(z) = 0.401z^{7195} + 0.599z^{\infty}$
10	Probability	$U(z) = 0.406z^{7516} + 0.594z^{\infty}$
	Time	$U(z) = 0.406z^{7219} + 0.594z^{\infty}$

number of iterations is set to 500. It means when the number of the cycles shown in Fig. 3 reaches 500, the algorithm finishes and returns the results. If we set this number to a larger value such as 1000 or 2000, the final outcome will be approximately the same as that for 500 iterations. Table 7 summarizes the results obtained from applying the proposed GA to schedule the workflow application to the grid resources in our case study. In Table 7, two different values, successful execution probability and mean service time, are considered for fitness values in the proposed GA, and pmf of the service time of the entire grid for the given workflow, as u-function, is presented for all valid values of parameter l ($l \in [1, 10]$).

As described in Section 6, and shown in Table 5, when the parameter l is set to *zero*, the proposed algorithm assigns all programs to a single resource which results in a single point of failure. To avoid this situation and fairly schedule the programs to the resources, this factor should be set to an appropriate value. For this reason, we exclude $l = 0$ from our results reported in Table 7. Furthermore, it can be concluded from Table 7 that assigning each of the programs to only a resource results in low successful execution probability. To overcome this shortcoming, each of the programs can be simultaneously assigned to two or more resources (redundancy in program assignment). In this case, the mean service time of an application may increase, but the successful execution probability will increase significantly. This is a trade-off between successful execution probability and service time.

B. The Comparison Study

Since methods in the state-of-the-art partially model some aspects of the system or evaluate different measures of interest, it is difficult to directly compare the numerical results of the proposed approach with the ones obtained from those previous methods. The comparison between a newly proposed method and previously presented ones is meaningful only if all assumptions made in the methods are common and the goals of the methods are the same. Otherwise, the complete comparison is almost impossible, and the methods can be only partially compared. Hence, in order to assess the proposed model and algorithms, we partly compare them with the method presented in [21] which is the most-related work in the state-of-the-art. However, there are some basic differences between our approach and the one presented in [21]. In the following, we mention the main differences and discuss the ways followed to allow the comparison of the approaches.

- The formalism used in [21] to model a grid resource is the Stochastic Activity Network (SAN), while we use SRN in this paper. Although the formalisms are similar, there are some differences between them and their supporting tools.
- The method proposed in [21] can only handle independent grid programs, named jobs in [21], whereas our proposed approach solves the problem when the grid programs/jobs have dependencies. Since our approach embraces a wide range of applications, for the sake of comparison, we reduce it and consider a subset of applications that only contains independent programs/jobs.
- The SAN model presented in [21] cannot compute the probability of successful execution of a given program/job on grid resources. The main goal of that SAN model is to compute the throughput of the resources in order to find the total makespan of a grid resource for a set of jobs. We can think of applying our method for computing the successful execution probability of a program, formulated by (1), (2), and (9), to the SAN model of [21], but it cannot help us to fully compare the methods, because the approach presented in [21] only considers independent programs/jobs as mentioned above. Therefore, in order to compute the successful execution probability of an application with n independent program, it suffices to only compute the probability of successful execution of each program/job, and then report the smallest number as the successful execution probability of

the entire application. This is one of the main reasons why we mention the approaches are *partly* compared herein since the approach presented in [21] does not handle dependent programs/jobs.

- The scheduling algorithm proposed in [21] uses Simulated Annealing (SA) to schedule grid jobs to the grid resources, while we use GA to schedule programs of an application to the grid resources. Although the representation of the problem and the methods used to reach to the near-optimal solution are different in these approaches, we can still compare them since the context of both optimization problems is the same.
- The aim of the scheduling algorithm presented in [21] is to find the best-possible scheduling string to minimize the overall makespan of the grid environment. However, the main goal of the scheduling algorithm presented in this paper is to find a suitable scheduling string to minimize the total service time of the overall workflow or maximize the successful execution probability of the workflow, which helps us to compute the pmf of service time of the entire grid for a workflow application. In order to find a common point between these two heuristics and partly compare them, the aim of both algorithms is considered to be minimizing the service time of the grid for a bag of independent programs/jobs, which is compatible with both approaches. To do this, we compute the service time of the grid environment, using the SAN model presented in [21], instead of computing the response time.

The grid environment considered in Section 7.A is used herein in the comparison study. The settings of the proposed model and algorithms are the same as those introduced in Section 7.A, and the setting of the SA-based scheduling algorithm is the same as that presented in [21]. There is no dependency among the programs/jobs which leads to the lower service time compared to the dependent programs/jobs discussed in Section 7.A. Fig. 4 shows the service time of the assumed grid environment resulted from both approaches. As can be seen in this figure, the service time of the environment for 1000 independent programs/jobs resulted from our approach is less than that of the method presented in [21], which shows the interest and the advantage of the approach proposed in this paper. The successful execution probability obtained from both approaches is the same since the programs/jobs are independent. It should be mentioned that the horizontal axis in Fig. 4 shows the threshold l which varies from 1 to 10. By increasing the value of l, more number of programs/jobs are assigned to low speed resources which leads to the increase of the service time.

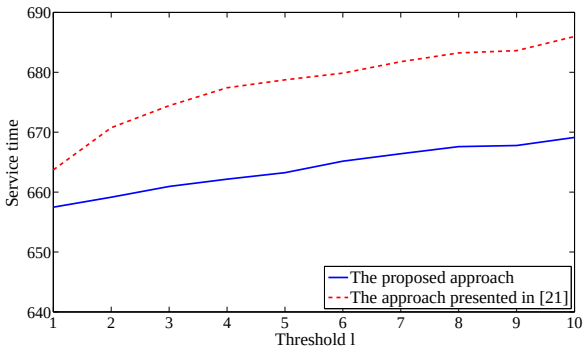

FIGURE 4. Comparison of the service time of the proposed approach and the method presented in [21] in a grid environment with 100 resources and an application with 1000 independent programs

8. CONCLUSIONS AND FUTURE WORK

In this paper, a new approach, based on SRNs, was proposed to assess the performability measures of a single grid resource. Afterward, the u-function technique was exploited to compute the service time and the successful execution probability of a program served by a resource. Using the proposed u-function, and the new composition operator on it, one can compute the service time of a grid resource for a set of programs. It leads to find the pmf of the service time of a grid for a workflow application consisting of dependent programs. Finally, a genetic-based scheduling algorithm was proposed to apply the outputs of the SRN model and the u-function in order to appropriately schedule programs of a workflow application to the resources distributed within the grid environment. The proposed scheduling algorithm could find a good enough scheduling string with the acceptable service time and successful execution probability. The applicability of the proposed approach to real grid systems was shown by solving detailed examples adopting the proposed approach.

There is a number of research issues remaining open for future work. In this paper, it is assumed that all resources are independent, having no interactions with each others. This simple topology, which is frequently used to relax the problem and make it easier to solve, can be replaced with a more complex and realistic one. For example, we can consider a tree structure (hierarchical connection) for a grid system, or consider a fully connected network in which all resources are connected to each other. If we take the structure of the network into account, we should change the u-function presented in this paper to be able to handle the common cause failures which have not been considered here. In addition to the structure of the environment, data transfer time can also be considered in the proposed methods. Although the programs studied in this paper are dependent on each

other, the time required to transfer data from a resource servicing a program to another resource servicing the subsequent program is not considered here. One can study the communication links among grid resources, and compute data transmission times according to the SP-graph representing the workflow application and topology of the grid environment. Moreover, failures of communication links between resources according to the structure of the system can be taken into account in future work.

Applying the proposed approach to the cloud computing environments and considering the specific characteristics of clouds (e.g. virtualization, rejuvenation, migration and so on) is also another idea for future work. Since cloud systems are widely used now-a-days, applying some modifications to the proposed approach can help us to be able to exploit it in analyzing cloud systems. Moreover, when considering a new environment, we can think about some important parameters in the new environment and try to compute the parameters using the proposed models and analyze them. For example, we can change the SRN model to be able to compute the power consumption measure in cloud systems by steady-state analyzing the SRN model.

Furthermore, studying various execution types of parallel subgraphs existing in a SP-graph is an interesting issue in this field. In our study, executing all programs of each parallel subgraph of a SP-graph is necessary for a workflow application to be executed. In some cases, executing only one path of a parallel subgraph suffices to start the execution of the subsequent subgraph(s) in the overall SP-graph. Generally, in these cases, we can consider the execution of only k paths of all n paths, $k \leq n$, of a parallel subgraph as a requirement for executing the entire subgraph. Moreover, general cases of program executions inside a workflow application can be studied by applying maximum and minimum functions to the execution time and/or successful execution probability resulting from all paths of a parallel subgraph. Considering each of these combinations requires defining a new operator for computing the corresponding u-function and may result in different scheduling strings.

REFERENCES

[1] Foster, I. and Kesselman, C. (2004) *The Grid 2: Blueprint for a New Computing Infrastructure*, second edition. Morgan Kaufmann, San Francisco, CA.

[2] Krauter, K., Buyya, R., and Maheswaran, M. (2002) A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, **32**, 135–164.

[3] SETI@home project. http://setiathome.ssl.berkeley.edu/. [Online: Accessed November 2016].

[4] Worldwide LHC Computing Grid (WLCG) project. http://wlcg.web.cern.ch/. [Online: Accessed November 2016].

[5] Li, M. and Baker, M. (2005) *The Grid Core Technologies*, first edition. John Wiley and Sons, Chichester, England.

[6] Entezari-Maleki, R. and Movaghar, A. (2012) A probabilistic task scheduling method for grid environments. *Future Generation Computer Systems*, **28**, 513–524.

[7] Levitin, G. and Dai, Y.-S. (2007) Service reliability and performance in grid system with star topology. *Reliability Engineering and System Safety*, **92**, 40–46.

[8] Entezari-Maleki, R. and Movaghar, A. (2010) A genetic-based scheduling algorithm to minimize the makespan of the grid applications. In Kim, T., Yau, S., Gervasi, O., Kang, B., Stoica, A., and Slezak, D. (eds.), *Grid and Distributed Computing, Control and Automation*, Communications in Computer and Information Science, **121**, pp. 22–31. Springer.

[9] Garg, R. and Singh, A. K. (2015) Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology, an International Journal*, **18**, 256–269.

[10] Tao, Y., Jin, H., Wu, S., Shi, X., and Shi, L. (2013) Dependable gridworkflow scheduling based on resource availability. *Journal of Grid Computing*, **11**, 47–61.

[11] Hao, Y. and Liu, G. (2014) Evaluation of nine heuristic algorithms with data-intensive jobs and computing-intensive jobs in a dynamic environment. *IET Software*, **9**, 7–16.

[12] Cordasco, G., Chiara, R. D., and Rosenberg, A. L. (2015) An AREA-oriented heuristic for scheduling DAGs on volatile computing platforms. *IEEE Transactions on Parallel and Distributed Systems*, **26**, 2164–2177.

[13] Wang, Y.-R., Huang, K.-C., and Wang, F.-J. (2016) Scheduling online mixed-parallel workflows of rigid tasks in heterogeneous multi-cluster environments. *Future Generation Computer Systems*, **60**, 35–47.

[14] Hu, M. and Veeravalli, B. (2013) Requirement-aware scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Transactions on Computers*, **62**, 2108–2114.

[15] Kaushik, A. and Vidyarthi, D. P. (2015) A green energy model for resource allocation in computational grid. *The Computer Journal*, **58**, 1530–1547.

[16] Azgomi, M. A. and Entezari-Maleki, R. (2010) Task scheduling modelling and reliability evaluation of grid services using coloured petri nets. *Future Generation Computer Systems*, **26**, 1141–1150.

[17] Balasangameshwara, J. and Raju, N. (2013) Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost. *IEEE Transactions on Computers*, **62**, 990–1003.

[18] Parsa, S. and Entezari-Maleki, R. (2012) Task dispatching approach to reduce the number of waiting tasks in grid environments. *The Journal of Supercomputing*, **59**, 469–485.

[19] Parsa, S. and Entezari-Maleki, R. (2012) A queuing network model for minimizing the total makespan of computational grids. *Computers and Electrical Engineering*, **38**, 827–839.

[20] Grzonka, D., Koodziej, J., Tao, J., and Khan, S. U. (2015) Artificial neural network support to monitoring

of the evolutionary driven security aware scheduling in computational distributed environments. *Future Generation Computer Systems*, **51**, 72–86.

[21] Entezari-Maleki, R., Bagheri, M., Mehri, S., and Movaghar, A. (2017) Performance aware scheduling considering resource availability in grid computing. *Engineering with Computers*, **33**, 191–206.

[22] Levitin, G. and Dai, Y.-S. (2008) Optimal service task partition and distribution in grid system with star topology. *Reliability Engineering and System Safety*, **93**, 152–159.

[23] Dai, Y.-S. and Levitin, G. (2007) Optimal resource allocation for maximizing performance and reliability in tree-structured grid services. *IEEE Transactions on Reliability*, **56**, 444–453.

[24] Entezari-Maleki, R. and Movaghar, A. (2011) Availability modeling of grid computing environments using SANs. *The 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2011)*, Split, Croatia, 15-17 September, pp. 1–6.

[25] Levitin, G., Dai, Y.-S., and Ben-Haim, H. (2006) Reliability and performance of star topology grid service with precedence constraints on subtask execution. *IEEE Transactions on Reliability*, **55**, 507–515.

[26] Dai, Y.-S. and Levitin, G. (2006) Reliability and performance of tree-structured grid services. *IEEE Transactions on Reliability*, **55**, 337–349.

[27] Entezari-Maleki, R., Trivedi, K., and Movaghar, A. (2015) Performability evaluation of grid environments using stochastic reward nets. *IEEE Transactions on Dependable and Secure Computing*, **12**, 204–216.

[28] Meyer, J. F. (1980) On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, **C-29**, 720–731.

[29] Smith, R. M., Trivedi, K. S., and Ramesh, A. V. (1988) Performability analysis: measures, an algorithm and a case study. *IEEE Transactions on Computers*, **37**, 406–417.

[30] Bruneo, D. (2014) A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, **25**, 560–569.

[31] Ghosh, R., Longo, F., Naik, V. K., and Trivedi, K. S. (2013) Modeling and performance analysis of large scale IaaS clouds. *Future Generation Computer Systems*, **29**, 1216–1234.

[32] Bruneo, D., Lhoas, A., Longo, F., and Puliafito, A. (2015) Modeling and evaluation of energy policies in green clouds. *IEEE Transactions on Parallel and Distributed Systems*, **26**, 3052–3065.

[33] Ushakov, I. (1987) Optimal standby problems and a universal generating function. *Soviet Journal of Computer Systems Science*, **25**, 79–82.

[34] Arsuaga-Rios, M. and Vega-Rodriguez, M. A. (2015) Multiobjective small-world optimization for energy saving in grid environments. *The Computer Journal*, **58**, 432–447.

[35] Entezari-Maleki, R., Sousa, L., and Movaghar, A. (2017) Performance and power modeling and evaluation of virtualized servers in IaaS clouds. *Information Sciences*, **394-395**, 106–122.

[36] Peterson, J. L. (1981) *Petri Net Theory and the Modeling of Systems*, first edition. Prentice Hall, Englewood Cliffs, N.J.

[37] Marsan, M. A., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G. (1995) *Modeling with Generalized Stochastic Petri Nets*, first edition. John Wiley and Sons, West Sussex, England.

[38] Marsan, M. A., Conte, G., and Balbo, G. (1984) A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, **2**, 93–122.

[39] Furfaro, A., Nigro, L., and Pupo, F. (2002) Distributed simulation of timed colored petri nets. *The 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications*, TX, USA, 11-13 October, pp. 159–166.

[40] Bause, F. and Kritzinger, P. S. (2002) *Stochastic Petri Nets: An Introduction to the Theory*, second edition. Vieweg+Teubner Verlag, Germany.

[41] Muppala, J. K. and Trivedi, K. S. (1992) Composite performance and availability analysis using a hierarchy of stochastic reward nets. In Balbo, G. and Serazzi, G. (eds.), *Computer Performance Evaluation, Modelling Techniques and Tools*, pp. 335–349. Elsevier Science Publishers B.V.

[42] Ciardo, G., Blakemore, A., ChimentoJr., P. F., Muppala, J. K., and Trivedi, K. S. (1993) Automated generation and analysis of Markov reward models using stochastic reward nets. In Meyer, C. D. and Plemmons, R. J. (eds.), *Linear Algebra, Markov Chains, and Queueing Models*, The IMA Volumes in Mathematics and its Applications, **48**, pp. 145–191. Springer.

[43] Ciardo, G., Muppala, J., and Trivedi, K. S. (1991) On the solution of GSPN reward models. *Performance Evaluation*, **12**, 237–253.

[44] Ciardo, G., Muppala, J. K., and Trivedi, K. S. (1989) SPNP: stochastic petri net package. *The 3rd International Workshop on Petri Nets and Performance Models (PNPM '89)*, Kyoto, Japan, 11-13 December, pp. 142–151.

[45] Jakoby, A., Liskiewicz, M., and Reischuk, R. (2001) Space efficient algorithms for series-parallel graphs. In Ferreira, A. and Reichel, H. (eds.), *18th Annual Symposium on Theoretical Aspects of Computer Science Dresden*, Lecture Notes in Computer Science, **2010**, pp. 339–352. Springer.

[46] Biton, O., Davidson, S. B., Khanna, S., and Roy, S. (2009) Optimizing user views for workflows. *The 12th International Conference on Database Theory*, St. Petersburg, Russia, 23-25 March, pp. 310–323.

[47] Sahner, R. A. and Trivedi, K. S. (1987) Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, **13**, 1105–1114.

[48] Wang, L., Siegel, H. J., Roychowdhury, V. P., and Maciejewski, A. A. (1997) Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, **47**, 1–15.

[49] Gephi: The Open Graph Viz Platform. https://gephi.org/. [Online: Accessed November 2016].

[50] The Failure Trace Archive. http://fta.scem.uws.edu.au/. [Online: Accessed November 2016].

[51] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/. [Online: Accessed November 2016].

[52] The Grid Workloads Archive. http://gwa.ewi.tudelft.nl/. [Online: Accessed November 2016].

[53] Caron, E., Garonne, V., and Tsaregorodtsev, A. (2007) Definition, modeling and simulation of a grid computing scheduling system for high throughput computing. *Future Generation Computer Systems*, **23**, 968–976.

[54] Valdes, J., Tarjan, R. E., and Lawler, E. L. (1982) The recognition of series parallel digraphs. *SIAM Journal on Computing*, **11**, 298–313.