

Performance Aware Scheduling Considering Resource Availability in Grid Computing

Reza Entezari-Maleki · Maryam Bagheri · Saeedeh Mehri · Ali Movaghar

Received: February 2016

Abstract This paper presents a mathematical model using Stochastic Activity Networks (SANs) to model a grid resource, and compute the throughput of a resource in servicing grid tasks wherein the failure-repair behavior of the processors inside the resource is taken into account. The proposed SAN models the structural behavior of a grid resource and evaluates the combined performance and availability measure of the resource. Afterwards, the curve fitting technique is used to find a suitable function fitted to the throughput of a resource for grid tasks. Having this function and the size of each grid job based on its tasks, an algorithm is proposed to compute the makespan of each available resource to a sequence of grid jobs assigned to the resource. Using the makespans of all grid resources computed in the previous step, the total makespan of the entire grid environment can be computed. Hence, a scheduling algorithm based on the Simulated Annealing (SA) meta-heuristic is presented to find a good enough scheduling of jobs on resources with the aim of minimizing the total makespan of the entire grid. Numerical results obtained

by applying the proposed SAN model, the algorithm presented to find the makespan of a single resource, and the proposed SA-based scheduling algorithm to a desktop grid show the applicability of the proposed approach in real grid environments.

Keywords Performance · availability · grid computing · stochastic activity network · curve fitting · simulated annealing.

1 Introduction

In pure performance evaluation, performance measures are assessed without any consideration of the fact that practically other dependability measures can affect the overall performance of the system. Especially, in highly distributed systems in which resources are most prone to fail, dependability of the components should be taken into consideration when the overall performance of the system is evaluated. Grid computing [14, 23] is one of the highly distributed systems which collects various resources from different administrative domains to use their processing capabilities in solving computational- and data-intensive problems in science and industry. In grids, resources are connected to a Grid Manager (GM) which receives grid jobs from grid users and dispatches them among the resources. The manager is responsible for sustaining the overall grid activities of scheduling and rescheduling [3, 25, 35]. Depending on the structure of the grid, GM can be considered as a single resource or even a distributed system composed of some resources [12, 15]. In grid computing environments, performance evaluation mainly focuses on completion time of the jobs submitted by grid and/or local users. In this respect, several related measures such as the total completion time of all jobs, expected waiting time of jobs,

Reza Entezari-Maleki
School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
E-mail: entezari@ipm.ir

Maryam Bagheri
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
E-mail: mbagheri@ce.sharif.edu

Saeedeh Mehri
Faculty of Management and Accounting, Allameh Tabataba'i University, Tehran, Iran
E-mail: s.mehri@atu.ac.ir

Ali Movaghar
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
E-mail: movaghar@sharif.edu

mean number of waiting grid jobs, expected sojourn time of jobs and so forth can be considered as performance measures [3, 12, 31, 32, 39].

In order to deliver jobs to the grid environment, grid resources should be available to be able to interact with users to get the jobs and service them. The availability of a grid resource can be affected by failure of the processor(s) inside the resource. When a resource fails, the GM has to use other available resources in the environment to schedule jobs on them [3, 25]. Therefore, the availability of resources highly influences the total time on which the grid can service a sequence of jobs. This time which is called the total *makespan* of the grid environment is a measure to assess the performance of the entire grid. The total makespan of a grid computing environment is defined as the largest makespans of grid resources. The makespan of a resource is the time slot between the start and completion of a sequence of jobs assigned to that resource [8, 11, 12, 18, 22, 30, 33, 36, 37, 39, 40]. Actually, grid is considered as a high throughput system, and minimizing the total makespan of the grid increases the throughput of the environment accordingly. In order to minimize the total makespan of the grid, assignment of jobs to grid resources should be done with more consideration. Hence, applying a suitable job scheduling algorithm to appropriately dispatch jobs to grid resources can help developers to increase the throughput of the environment.

Since decision about the assignment of jobs to resources and finding the best match between jobs and resources is an NP-complete problem [2, 8, 11, 12, 18, 22, 24, 30, 33, 36, 39, 40, 41], we use Simulated Annealing (SA) meta-heuristic [2, 8, 19, 24, 33, 40, 41] to propose a job scheduling algorithm to solve the problem. SA is a generalization of the Monte Carlo method for statistically finding the global optimum for multivariate functions. The concept originates from the way in which crystalline structures are brought to more ordered states by an annealing process of repeated heating and slowly cooling the structures. For more information on SA in job scheduling context please see [2, 8, 24, 33, 40, 41]. Our proposed SA-based scheduling algorithm tries to minimize the total makespan of the grid by appropriately assigning grid jobs to grid resources. The most important part of the proposed algorithm is using a more practical algorithm to find the makespan of each grid resource in which the makespan of a resource is calculated using a mathematical solution which estimates the throughput of the resource for grid jobs consisting of many grid tasks in each time instant. To do this, we first model a grid resource using Stochastic Activity Networks (SANs) [27, 28, 29], and then, find the function fitted to the results obtained by analyzing the SAN

model. The proposed SAN models the arrival of grid and local tasks to a resource, and the servicing process of them in the resource. Moreover, the failure and repair behavior of the processors inside the resource are modeled using the proposed SAN. After modeling a grid resource by a SAN, some important measures can be assessed using the transient and steady state analysis of the proposed SAN. Therefore, the performance of a single grid resource can be evaluated wherein the availability of its processors is taken into account.

The main contribution of this paper can be summarized into three categories. First, the paper uses the SAN formalism to formally model a single grid resource and compute the combined performance and availability measure of the resource. The proposed model not only considers the simultaneous execution of grid and local tasks, but also it applies higher execution priority to local tasks against grid ones. Since in grid environments, local tasks are generally preferred to be serviced before grid tasks [14], we model this real assumption using specific characteristics of SANs which are explained in the following sections with details. In addition to modeling a grid resource by SAN formalism, we define some interesting output measures on the proposed SAN to be able to find the combined performance and availability measure. It turns out that each time we construct the SAN model, it is required to know the status of available resources to be able to compute the performability measure of a grid resource. Since this computation does not take more time, it can be done immediately in the GM after resource discovery phase. Second, the paper proposes an algorithm to find the makespan of a single grid resource for a sequence of grid jobs. After analyzing the SAN model and finding the throughput of a single grid resource for grid tasks, we use the curve fitting technique to find the most fitted function to the results obtained from the numerical analysis of the SAN model. The function fitted to the throughput of a grid resource is used to find the makespan of the resource for a possible scheduling of jobs on the resource. Third, we use the proposed SAN model and the algorithm proposed to find the makespan of a single grid resource to present a scheduling algorithm to dispatch grid jobs to grid resources with the aim of minimizing the total makespan of the grid environment. The proposed job scheduling algorithm uses SA meta-heuristic to find the most suitable schedule of grid jobs on grid resources.

The remaining parts of the paper are organized as follows. Section 2 introduces some related work done in this research area. Section 3 provides a short overview on SANs. Section 4 presents the proposed approach which is composed of four subsections; a SAN model

for a single grid resource, methods for computing the measures of interest, an algorithm to find the makespan of a single grid resource, and a SA-based job scheduling algorithm to minimize the total makespan of the grid environment. In Section 5, a simple example on a hypothesis grid environment is presented to show how the proposed approach can be used. Moreover, a sensitivity analysis on makespan of a grid resource when the internal configuration of the resource changes is done in Section 5. In Section 6, numerical results obtained by applying the proposed approach to two different grid environments are reported to show how the proposed approach can be applied to real systems. Finally, Section 7 concludes the paper and presents some guidelines for future work.

2 Related Work

Since our proposed approach includes two main parts; mathematically model and scheduling algorithm, we organize the related work section in two different parts called *performance modeling* and *task scheduling*.

2.1 Performance Modeling

There have been proposed many analytical models to evaluate the performance and availability of distributed systems. In the following, some of the related work done in the field of performance and availability/reliability evaluation in distributed computing systems are introduced.

Entezari-Maleki et al. [13] have proposed three Stochastic Reward Net (SRN) models to evaluate the performability of a single grid resource, and then, applied the single models to model and evaluate the whole grid environment. Since the exact monolithic model of an entire grid shows state space explosion, two approximate models were proposed in [13] to estimate the performability of whole grid environment. Entezari-Maleki et al. [9] have proposed a Markov Reward Model (MRM) to model and evaluate the performability of a single grid resource. Although MRM presented in [9] is a mathematical solution which models a grid resource appropriately, it still ignores some details of the resource such as existing various numbers of processors inside the resource and failing the processors servicing grid and local tasks which can be seen in real systems. None of the approaches presented in [9] and [13] discusses about using the resulting measures in optimizing the system to reach a better performability.

Longo et al. [26] have proposed an SRN model to analyze the availability of large scale IaaS cloud. The

model was first presented in its monolithic form in one level, and then, it was decomposed to overcome largeness problem suffered by the monolithic model. The model presented in [26] can be appropriately used to analyze the availability of hot, warm and cold pools of resources in a cloud, and finally, analyze the total availability of an IaaS cloud. Another model in the same context has been proposed in [17] by Ghosh et al. In [17], interacting stochastic sub-models were presented to evaluate the performance of large scale IaaS clouds while workload of the cloud, system characteristics and management policies were taken into account. Entezari-Maleki et al. [10] have proposed a SAN to model and evaluate the availability of a grid environment. The SAN model presented in [10] can assess the availability of grid management system together with availability of grid resources. Furthermore, the impact of applying different task scheduling policies to dispatch grid and local tasks among the grid resources can be appropriately studied by the SAN model presented in [10]. It should be mentioned that all models proposed in [10, 17, 26] only evaluate the availability of the system paying no attention to the performance measures.

Parsa et al. [31, 32] have proposed Queuing Network (QN) and Generalized Stochastic Petri Net (GSPN) solutions to model and evaluate the performance of grid computing systems. The performance measures evaluated in [31] and [32] are the total makespan of the environment and the mean number of grid tasks waiting to receive service from the environment, respectively. Both QN and GSPN models proposed in [32] only consider grid tasks submitted by grid users paying no attention to the local tasks of the system, whereas the models proposed in [31] consider both grid and local tasks. Difficulty with all of the models proposed in [31] and [32] is that the models can only compute pure performance measures of the grid environment, and they do not handle the situation in which one or more of the resources fail. Azgomi et al. [3] have presented a Colored Petri Net (CPN) model to show the workflow of task execution in grid computing, and compute the reliability of a grid service. Although the CPN model proposed in [3] precisely investigates the failure of grid resources and its effect on the performance of resources, it ignores considering local tasks and solves the problem when a single resource is considered in isolation without having any interaction with others.

2.2 Task Scheduling

Many scheduling algorithms can be found in literature which dispatch the requests to the servers in distributed systems. In the following, some of the research papers

proposing new scheduling schemes in distributed computing systems are introduced.

Garg et al. [16] have proposed an adaptive method to schedule dependent tasks to grid resources where the availability of computing nodes and links are considered to be variable due to the existence of local load. Kolodziej et al. [22] have defined the problem of independent batch scheduling in computational grids as a three-objective global optimization problem with makespan, flow-time and energy consumption minimized according to different security constraints. The optimization problem defined in [22] was solved by genetic based meta-heuristics. Entezari-Maleki et al. [12] have proposed a probabilistic task scheduling method in grids to decrease the mean response time of tasks in grid computing environments. The method used in [12] to evaluate the mean response time of entire grid for servicing grid tasks is Discrete Time Markov Chain (DTMC) which models each of the clusters of the grid. However, the model presented in [12] still ignores the failure-repair behavior of grid resources, and the performance is evaluated purely without any consideration about dependability issues. The same authors have proposed a genetic based scheduling algorithm to minimize the makespan of a grid environment, and thereby, maximize the throughput of entire grid environment [11]. The genetic algorithm proposed in [11] applies a good representation of the problem in the form of chromosomes to help the algorithm to converge to the suitable result in situations that the numbers of tasks and resources are very large.

Parsa et al. [30] have proposed a task scheduling algorithm called RASA which uses the advantages of both Min-Min and Max-Min algorithms and covers their disadvantages simultaneously. To achieve this, RASA firstly estimates the completion time of the tasks on each of the available resources, and then, applies the Min-Min and Max-Min algorithms alternatively. Damodaran et al. [8] have proposed a SA algorithm to minimize the makespan of a group of processing machines working in parallel. In the algorithm proposed in [8], each machine can simultaneously process several jobs as a batch. Random instances were used to compare the results of the proposed SA in [8]. Abdulal et al. [2] have presented a SA-based scheduling algorithm to find a good solution for scheduling independent tasks in grid environments. The measures considered in the SA algorithm presented in [2] are makespan, time to release, reliability and flow-time. Torabzadeh et al. [40] have proposed a job scheduling algorithm based on SA meta-heuristic. The aim of the algorithm presented in [40] is minimizing the weighted sum of makespan and the mean completion time for n available jobs.

Rathore et al. [34] have provided an extensive survey of the existing load balancing and job migration techniques in grid environments together with a detailed classification of the existing techniques based on different parameters such as structures, scheduling strategies, techniques, algorithms, attributes, models, components, performance metrics, and challenges. After analyzing the previously proposed methods, a new approach has been devised in [34] which uses the hierarchical load balancing technique and fault tolerance with check-pointing based job migration technique. The same authors have proposed a novel technique to tackle hierarchical load balancing in grid environments while maintaining the resource utilization and response time for dynamic and decentralized grids [35]. The proposed method was analyzed based on variable threshold value. To achieve this, the load was divided into different categories, such as lightly loaded, under-lightly loaded, overloaded, and normally loaded. It is worthwhile to mention that the method proposed in [35] eliminates the scalability complexity of a site by a dynamic threshold value, while most of the existing load balancing techniques use a static threshold value which causes problems in a large-scale grid environment.

Other related methods and algorithms in this research field can be found in the literature. Generally, each of the methods presented in this area has its own advantages and disadvantages. There are some problems with the previously proposed methods in grid context. One of the problems is that only a few of them consider both local and grid tasks, and simultaneous execution of them inside grid resources. Some papers that consider both types of tasks only compute pure performance of the grid environment paying no attention to the failure-repair behavior of resources. Proposing only a framework to evaluate the performance and/or dependability of grids disregarding the scheduling problem of jobs on grid resources is another difficulty in some previous research papers. Moreover, the scheduling algorithms proposed in this area only optimize a fitness value which is a simple criterion or combination of some simple criteria without proposing a general framework and formal way to compute the fitness value compatible with the system under study. We try to address the problems mentioned above in our proposed approach in this paper.

3 Overview of SANs

In order to compute the combined performance and availability of a single grid resource, SANs are used in this paper. For the sake of brevity, only the basic concepts of SANs are presented in this section, but for

more information on formal definition, structure, and characteristics of SANs please see [4, 27, 28, 29, 38].

SANs are stochastic generalization of PNs, generally defined for the modeling and analysis of distributed real-time systems. SANs are more powerful and flexible than other stochastic extensions of PNs such as Stochastic Petri Nets (SPNs) and Generalized Stochastic Petri Nets (GSPNs). In General, SANs are probabilistic extensions of Activity Networks (ANs) which have been equipped by a set of activity time distribution functions, reactivation predicates and enabling rate functions. The nature of the extension is similar to the extension that constructs SPNs from classical PNs. Informally, SANs can be described with three major primitives called *places*, *activities* (timed and instantaneous), and *gates* (input and output). The graphical notations of these primitives are shown in Fig. 1 [4, 28, 29, 38]. Followings are descriptions of each of SAN elements [4, 28]:

- *Place*: it is similar to a place in PNs and is shown by a circle. Places can act as depositories which tokens are put in. For example, a place in a SAN model can represent an input buffer of the specified size.
- *Timed activity*: timed activities represent activities of the modeled system whose durations impact the system's ability to perform. A timed activity has m inputs and n outputs, where $m + n > 0$. Each of the inputs of a timed activity can be a place or an input gate, and also each of the outputs can be a place or an output gate. An activity distribution function, enabling rate and n -ary computable predicates called the reactivation predicates are associated for each timed activity. Timed activities are shown by thick bars or boxes.
- *Instantaneous activity*: instantaneous activities represent the system activities which can be completed in a negligible amount of time. An instantaneous activity has m inputs and n outputs. A case probability function (i.e. a computable partial function that returns a value in range $[0, 1]$) is associated for each instantaneous activity. Case probabilities associated with instantaneous activities permit the realization of uncertainty. Instantaneous activities are shown by thin bars.
- *Input gate*: an input gate has a finite set of inputs and one output. For each of the input gates an n -ary computable predicate called the enabling predicate and a computable partial function called the input function are associated. Input gates are shown by the filled triangles.
- *Output gate*: an output gate has a finite set of outputs and one input. A computable function called the output function is associated to each such out-






Element	Place	Timed activity	Instantaneous activity	Input gate	Output gate
Graphical notation					

Fig. 1 The graphical representation of SAN elements

put gate. Output gates also are shown by the filled triangles.

Modeling and analysis with SANs need a software tool to help construct and evaluate the model. The original definition of SANs has been used as a modeling formalism in some modeling tools, such as METASAN, UltraSAN and Möbius [4, 7]. All of these tools are intended for the evaluation of operational aspects (such as performance, dependability, and performability) of systems. In this paper, the Möbius tool [7] is used to construct and analyze the proposed SAN model.

4 The Proposed Approach

This section explains the proposed approach with details in four subsections. In Subsection 4.1, the SAN model proposed to evaluate the combined performance and availability of a single grid resource is presented. Subsection 4.2 introduces some measures which can be obtained by analyzing the proposed SAN model. The algorithm proposed to compute the makespan of a single grid resource for a sequence of grid jobs is given in Subsection 4.3, and finally, the SA-based job scheduling algorithm is presented in Subsection 4.4.

4.1 The Proposed SAN Model

Fig. 2 shows the SAN model of a single grid resource wherein the failure-repair behavior of its processors is taken into account. Input parameters of the proposed SAN are: grid and local queue sizes of the resource (M_G and M_L), grid and local tasks arrival rates (λ_G and λ_L), number of the processors inside the resource (N), service rate of each processor (μ), failure rates of idle and busy processors (γ_i and γ_b), and repair rate of a failed processor (δ).

Places P_{UP} and P_{DW} represent the *up* (operational) and *down* (non-operational) processors inside the resource, respectively. It is assumed that there are N operational homogeneous processors in the resource when it starts to service the tasks, so there are N tokens in place P_{UP} in the beginning. Timed activity TA_{JPF} represents the failure process of idle processors. It should

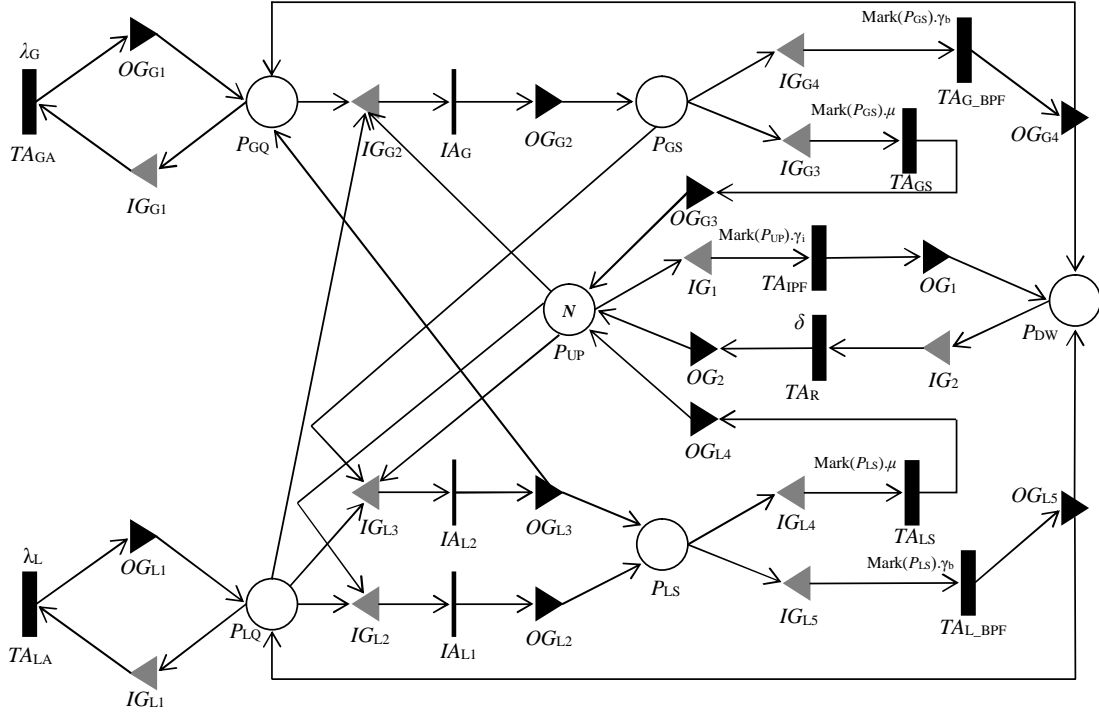


Fig. 2 The proposed SAN model for a single grid resource

be noted that the times assigned to all timed activities follow exponential distribution which is an acceptable assumption in this area [3, 9, 10, 13, 17, 25, 26, 31, 32]. The rate of timed activity T_{AIPF} is $Mark(P_{UP}) \cdot \gamma_i$ where $Mark(P_{UP})$ denotes the number of tokens inside place P_{UP} and γ_i is the failure rate of an idle processor. Timed activity T_{AR} represents the repair process of a failed processor. The completion rate of this activity is δ which shows that there exists only a repair facility in the system because its rate is not marking dependent.

As shown in Fig. 2, there are two separate lines of task arrivals in the proposed SAN. The first line which is initiated with timed activity T_{AGA} models the arrival of grid tasks to the resource. When activity T_{AGA} is completed with rate λ_G , a token is deposited into place P_{GQ} to show that a grid task has already been submitted to the resource, and it is waiting to get service from one of the idle operational processors. The completion capability of timed activity T_{AGA} is checked by input gate IG_{G1} to prevent this activity from completion if there are M_G tokens inside place P_{GQ} where M_G denotes the grid queue size of the resource. If there is a waiting grid task in the grid queue of the resource (one token in place P_{GQ}), and at least one operational processor (one token in place P_{UP}), instantaneous activity I_{AG} can be completed. This condition is checked by input gate IG_{G2} . Upon completion of instantaneous

activity I_{AG} , a token from place P_{GQ} together with another token from place P_{UP} is removed and a token is deposited into place P_{GS} to show that a grid task is servicing by one of the processors. These adding/removing are done by output gate OG_{G2} and input gate IG_{G2} .

Existence of a token in place P_{GS} causes two timed activities T_{AGS} and T_{AG_BPF} to be enabled. Timed activity T_{AGS} models servicing process of a grid task inside the resource with completion rate $Mark(P_{GS}) \cdot \mu$, where $Mark(P_{GS})$ denotes the number of tokens inside place P_{GS} and μ is the service rate of a processor. After completion of timed activity T_{AGS} , a token is deposited into place P_{UP} to show that a processor has already serviced a grid task, and it is ready to be assigned to another task submitted to the resource. Timed activity T_{AG_BPF} models the failure process of a busy processor which services a grid task. The completion rate of this activity is also marking dependent which multiplies the failure rate of a single busy processor (γ_b) by the number of tokens inside place P_{GS} . After completion of timed activity T_{AG_BPF} , a token is put into place P_{DW} to show that a processor has failed, and it should be repaired before being able to service other grid/local tasks. Moreover, the failed grid task is returned to the queue to be scheduled on another processor later.

The same configuration is considered for local tasks in which the second line of task arrivals is initiated

Table 1 Input predicates/functions of some input gates of the SAN model presented in Fig. 2

Gate	Predicate	Function
IG_{G1}	$Mark(P_{GQ}) < M_G$;
IG_{G2}	$(Mark(P_{GQ}) > 0)$ and $(Mark(P_{UP}) > 0)$ and $(Mark(P_{LQ}) == 0)$	$Mark(P_{GQ}) = Mark(P_{GQ}) - 1;$ $Mark(P_{UP}) = Mark(P_{UP}) - 1;$
IG_{L1}	$Mark(P_{LQ}) < M_L$;
IG_{L2}	$(Mark(P_{LQ}) > 0)$ and $(Mark(P_{UP}) > 0)$	$Mark(P_{LQ}) = Mark(P_{LQ}) - 1;$ $Mark(P_{UP}) = Mark(P_{UP}) - 1;$
IG_{L3}	$(Mark(P_{LQ}) > 0)$ and $(Mark(P_{UP}) == 0)$ and $(Mark(P_{GS}) > 0)$	$Mark(P_{LQ}) = Mark(P_{LQ}) - 1;$ $Mark(P_{GS}) = Mark(P_{GS}) - 1;$

with timed activity TA_{LA} with rate λ_L modeling the arrival of local tasks to the resource. For the sake of brevity, the explanations about the components (places, timed/instantaneous activities, and input/output gates) related to the local tasks are not presented here, but they are exactly the same as those for the corresponding components of grid tasks. The important point which should be mentioned here is that the local queue size of the resource is M_L which is checked by input gate IG_{L1} to make sure that the number of local tasks submitted to the resource does not exceed this threshold. Moreover, since a grid resource voluntarily joins the grid environment, it services local tasks submitted by local users in its administrative domain with higher priority over grid tasks [14]. To model this, two types of priority disciplines, non-preemptive and preemptive, can be considered. The model shown in Fig. 2 models the preemptive priority discipline in which a newly arriving local task preempts a servicing grid task if there is no idle processor to be allocated to the local task. This is controlled by input gates IG_{G2} and IG_{L3} . The predicate and input functions of these gates can be seen in Table 1. As can be seen in this table, input gate IG_{G2} not only checks the existence of a token in both places P_{GQ} and P_{UP} as mentioned earlier, but also it checks the number of tokens inside place P_{GQ} to be *zero*. Using this mechanism, we can assign higher execution priority to local tasks over grid ones. However, the preemptive priority is modeled by input gate IG_{L3} . As can be seen in Table 1, the condition $[Mark(P_{LQ}) > 0$ and $Mark(P_{UP}) == 0$ and $Mark(P_{GS}) > 0]$ is checked in the predicate part of input gate IG_{L3} . If this predicate is evaluated to true, the instantaneous activity IA_{L2} is completed, and a token is put into place P_{LS} to show that the local task has taken a processor from a servicing grid task. Moreover, a token is deposited into place P_{GQ} upon completion of activity IA_{L2} to show that the preempted grid task has been resubmitted to the waiting grid queue to be executed later.

Although the predicates and functions of input gates introduced above exactly implement the aforementioned mechanisms in the network, to provide a better understanding of the proposed SAN, the predicates and functions of input gates IG_{G1} , IG_{G2} , IG_{L1} , IG_{L2} , and IG_{L3} are presented in Table 1. It should be mentioned that we use a simple notation for representing the predicates and functions to increase the readability of them which is different from the language of Möbius. The input gates which have not been listed in Table 1 are simple input gates which check the existence of a token in their input places, and remove a token from them when their corresponding activities are completed. Similarly, all output gates shown in Fig. 2 are simple output gates which add a token to their output places whenever their corresponding activities are completed.

4.2 Measures of Interest

The model presented in Subsection 4.1 can be used to compute various performance and dependability measures by assigning appropriate reward rate to each feasible marking of the network. Let r_i denote the *reward rate* assigned to marking i of the SAN model presented in Fig. 2. If π_i denotes the steady state probability for the SAN to be in marking i , then the expected steady state reward rate can be computed as $\sum_i r_i \pi_i$. An *impulse reward* is the instantaneous reward gained when firing activity t in a given marking [6]. In this paper Möbius tool [7] is used to solve the numerical examples of the proposed SAN model and obtain performance measures. The interesting measures we are intending to compute them by analyzing the proposed SAN are: (1) the *Throughput* of a resource for grid tasks, (2) the *Mean Response Time* of a resource for grid tasks, and (3) the *Failure Probability* of grid tasks. In the following, we describe each measure and its calculation method.

Throughput of a resource for grid tasks. It is a measure of how many grid tasks a resource can process in a given amount of time. In the steady state, the throughput of a resource for grid tasks is equal to the effective arrival rate of grid tasks to the resource. Let $\pi(P_{GS} = i)$ denote the steady state probability of there being i tokens in place P_{GS} . Hence, the throughput of a resource for grid tasks can be computed by Equation (1).

$$\text{Throughput} = \sum_{i=1}^N \pi(P_{GS} = i) \cdot i \cdot \mu, \quad (1)$$

where N and μ denote the total number of processors inside the resource and service rate of each processor, respectively.

Failure Probability of grid tasks (P_F). It is the probability that a processor servicing a grid task fails. Using the concept of impulse reward, we can compute this probability by measuring the reward obtained when firing timed activity TAG_{BPF} . In Möbius, there is a direct way to compute the impulse reward of each activity, so we use this capability of Möbius.

Mean Response Time of a resource for grid tasks ($E[R]$). It is the mean time from a grid task is submitted to a resource until it has been processed by the resource which is the time from the instant that a token is deposited into place P_{GQ} until it is removed from place P_{GS} . Using Little's law [5], the steady state mean response time of grid tasks, $E[R]$, can be computed as Equation (2).

$$E[R] = \frac{E[\#P_{GQ}] + E[\#P_{GS}]}{\lambda_{eff}} \quad (2)$$

where $E[\#P_i]$ is the mean number of tokens in place P_i , which is computed by assigning reward rate $\#P_i$ to the states of the underlying Markov chain of the proposed SAN. Moreover λ_{eff} is the effective arrival rate of grid tasks which is computed by Equation (3).

$$\lambda_{eff} = (1 - P_b)\lambda_G \quad (3)$$

where λ_G is the arrival rate of grid tasks, and P_b is the blocking probability of grid task arrivals which is computed by assigning the following reward to the proposed SAN model.

$$r_i = \begin{cases} 1, & E[\#P_{GQ}] \geq M_G \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where M_G is the grid queue size of the resource.

4.3 The Proposed Algorithm to Find the Makespan of a Single Grid Resource

Modeling a grid resource by the proposed SAN and using the methods introduced in Subsection 4.2, we can compute the throughput a resource for grid tasks. As mentioned earlier, the throughput of a resource is the number of tasks processed over a given interval of time. Using the curve fitting technique [20], we can find a most suitable function fitted to the results obtained by analyzing the proposed SAN model. Let $f_i(t)$ denote the function fitted to the results of the throughput analysis of grid resource R_i for grid tasks. Suppose scheduling string S represents a sequence of grid jobs assigned to all grid resources in the environment, and the sequence Seq inside scheduling string S shows the grid jobs assigned to resource R_i . Let grid job J_j denote the first job in sequence Seq assigned to resource R_i to be executed. If the size of job J_j is $Size_j$ tasks, then the time required for servicing job J_j by resource R_i will be equal to t_1^{ij} which is computed by solving Equation (5).

$$\int_{t_0^{ij}}^{t_1^{ij}} f_i(t) dt = Size_j \quad (5)$$

where t_0^{ij} is the ready time of resource R_i for servicing job J_j which should be set to a fixed number when the resource starts to execute the job. In the beginning in which the resource starts to service grid jobs, this number is set to *one*. After finding t_1^{ij} which shows the time required for servicing grid job J_j by resource R_i , the ready time t_0^{ik} should be updated to t_1^{ij} before being able to allocate R_i to the next grid job named J_k ($t_0^{ik} = t_1^{ij}$). Therefore, the time interval $[t_0^{ij}, t_1^{ij}]$ is the time required for resource R_i to service grid job J_j , which is the first job assigned to R_i in sequence Seq . With the same procedure, time interval $[t_0^{ik}, t_1^{ik}]$ shows the time required to service job J_k by resource R_i while J_k is the second job in scheduling sequence Seq assigned to resource R_i after job J_j . Using the procedure mentioned above, we can compute the maximum time taken by each resource to service a sequence of grid jobs assigned to that resource in a given scheduling string by Equation (6).

$$\text{Makespan}_i = t_1^{il} \quad (6)$$

where Makespan_i denotes the makespan of resource R_i and J_l is the last job assigned to resource R_i in sequence Seq under scheduling string S . The aforementioned procedure is summarized in Algorithm 1.

Algorithm 1: The proposed algorithm to find the makespan of grid resource R_i under scheduling string S

Input: sequence $Seq = \{J_1, J_2, \dots, J_l\}$ showing the grid jobs assigned to resource R_i in scheduling string S and the size of job J_j as $Size_j$ for $1 \leq j \leq l$

Output: makespan of resource R_i as $Makespan_i$

```

1  $t_0^{i1} \leftarrow 1$ 
2 Model resource  $R_i$  using the SAN model presented in Fig. 2
3 Compute the throughput of resource  $R_i$  for grid jobs using Equation (1)
4 Find the function  $f_i(t)$  fitted to the results obtained from Equation (1)
  for  $j = 1$  to  $l$  do
5   Compute  $t_1^{ij}$  using Equation (5)
6    $t_0^{i\{j+1\}} \leftarrow t_1^{ij}$ 
  end
7 Return  $t_1^{il} = t_0^{i\{l+1\}}$  as  $Makespan_i$ .
```

4.4 The Proposed SA-based Scheduling Algorithm

Using the algorithm presented in Subsection 4.3, we can compute the total makespan of a grid environment in servicing grid jobs. For this reason, a sequence of n independent grid jobs is assumed to be submitted to the grid environment to be serviced. Assume a possible scheduling of n grid jobs on m grid resources named S . Applying the algorithm shown in Algorithm 1 to each grid resource, the makespan of the resource in servicing grid jobs assigned to it can be computed. Having this measure for all resources, we can compute the total makespan of the entire grid environment, called *Makespan*, using Equation (7).

$$Makespan = \max_i Makespan_i \quad (7)$$

where $1 \leq i \leq m$, and m is the number of all grid resources available in the environment. Changing the scheduling string S results in different makespans for the grid environment. Since our aim is minimizing the measure *Makespan*, we present a SA-based scheduling algorithm to appropriately dispatch grid jobs to grid resources to minimize this measure.

The most important part in the application of SA is the generation of the initial solution, and creation of a set of neighbors. To generate an initial solution, a representation of the problem should be defined. In this problem, we use an integer encoding in which each possible solution is represented by a vector of integers.

For a problem with n jobs and m resources, the length of the possible solution is n and each integer number can take a value between 1 and m , where each number represents the resource which is allocated to the corresponding job. As an example, if we have *four* jobs ($n = 4$) and *two* resources ($m = 2$) in the environment, a possible scheduling string can be as $S = 1221$ which assigns jobs J_1 and J_4 to resource R_1 , and jobs J_2 and J_3 to resource R_2 . It is worthwhile to mention that although assigning all jobs to a single resource (the fastest resource here) is a valid solution for the problem, we avoid this situation because it generates a single point of failure in which failing a resource causes the execution of all jobs fails. Therefore, to generate more balanced and fair scheduling, we define a *low threshold* factor called l for a possible solution to represent the minimum number of jobs assigned to each resource in a valid solution. It turns out that the maximum value for low threshold is $l = \lfloor \frac{n}{m} \rfloor$. In the example mentioned above with *four* jobs and *two* resources, the low threshold factor can take values 0 to 2 ($l = 0, 1, 2$). We avoid $l = 0$, which generates a single point of failure, but $l = 1$ and $l = 2$ are two valid values for l in scheduling strings of this example. For the scheduling string mentioned above ($S = 1221$), l is 2, and for a new scheduling string named $S' = 1121$, which can be generated from S , the low threshold factor is 1, which shows that for each resource it has been assigned at least *one* job.

In order to generate a neighbor of a solution namely a *new solution*, a simple modification on the old solution is done in which a random cell of the old solution is selected and its value is exchanged with a random number in range $\{1, 2, \dots, m\}$, where m is the number of resources in the grid environment. We name this modification as *exchange method* in the proposed algorithm. In exchange method, we randomly change the resource allocated to execute a randomly selected job. It is worthwhile to state that the exchange method should preserve the solution validity satisfying the low threshold factor in each step. As an example, consider the scheduling string $S = 12211$ wherein $n = 5$ and $m = 2$. In the first step, we should choose a random number in range $\{1, \dots, n\}$ ($R_1 \in \{1, 2, \dots, 5\}$), since the solution contains 5 cells. Afterwards, another random number in range $\{1, \dots, m\}$ ($R_2 \in \{1, 2\}$) should be selected. Let $R_1 = 3$ and $R_2 = 1$ denote the first and second random numbers generated for this reason. According to the exchange method explained above, we should change the 3rd number of solution $S = 12211$ with number 1, which generates a new solution $S' = 12111$. As mentioned earlier, the exchange method should preserve the validity of the new solution. For example, if we consider the low threshold factor as $l = 1$, the newly generated schedul-

ing string $S' = 12111$ will be a valid solution, but if we consider $l = 2$, it will be an invalid solution. In this situation ($l = 2$), $R_1 = 4$ and $R_2 = 2$ for example, can generate a new valid solution as $S'' = 12221$.

After generating a new solution (S'), its makespan is computed using Algorithm 1 and Equation (7) as done for old solution (S). To do this, all grid jobs assigned to resource R_i , $i \in \{1, \dots, m\}$, in the new solution are specified, and then, the measure $Makespan_i$ is obtained by applying Algorithm 1 to resource R_i . After finding makespans of all resources ($Makespan_i, \forall i \in \{1, \dots, m\}$), the total makespan of the new solution is computed using Equation (7). Now, having both makespans, $Makespan(S)$ and $Makespan(S')$, the acceptance test can be applied. The acceptance test for a solution in each step works as follows: if the new solution is better than the old one, then the proposed SA-based algorithm will replace it, if it is worse, the algorithm replaces it with probability P . The probability P depends on the difference between the total makespans of old and new solutions, and the control parameter T named temperature. It is expected that the probability of moving to the new solution decreases when the difference between two total makespans gets magnified (the total makespan of new solution gets worse). On the other hand, the temperature parameter plays a crucial role in controlling the evolution process in which cooling the temperature slowly causes the uphill movements becomes less and less as the run progresses [24,41]. Therefore, we need a formula to decrease probability P by increasing the difference between total makespans of old and new solutions, and decreasing the temperature parameter T . The exponential function as $P = \exp\left(\frac{Makespan(S) - Makespan(S')}{T}\right)$ where $Makespan(S) < Makespan(S')$ can help us to reach this goal, but generally, any function which satisfies aforementioned conditions can be used. It is worthwhile to mention that the mechanism of replacing a good solution with a worse solution in the proposed SA-base algorithm with small probability P is done to prevent the algorithm to get stuck in local optimal points, and provide it with the chance of achieving global optimal solution. However, it should be done very intelligently to be able to finally reach a good enough solution in a timely manner.

This procedure is continued until the termination condition is satisfied. In our proposed algorithm, the number of iterations is used to specify the termination condition, but in general, other criteria such as allocation constraints (e.g. time required to run the algorithm), difference between makespans of two consecutive runs, manual inspection, combination of the conditions and so forth can be used as the termination con-

Algorithm 2: The proposed SA-based scheduling algorithm to minimize the total makespan of a grid environment

```

1 Generate a random initial solution named  $S$  according
  to the representation method
2  $FinalS \leftarrow S$ 
3  $FinalMakespan \leftarrow Makespan(S)$ ,
  where  $Makespan(S)$  computes the total makespan of
  the environment under scheduling string  $S$  using
  Algorithm 1 and Equation (7)
4  $i \leftarrow 0$ 
5  $t_i \leftarrow 10000$ 
  while (the termination condition is not met ) do
6   Generate new solution  $S'$  using exchange method
7   Calculate  $Makespan(S')$ 
8    $P \leftarrow \exp\left(\frac{Makespan(S) - Makespan(S')}{t_i}\right)$ 
9    $r \leftarrow random[0,1]$ 
  if ( $Makespan(S') < FinalMakespan$ ) or
  ( $r < P$ ) then
10     $S \leftarrow S'$ 
11     $FinalS \leftarrow S'$ 
12     $FinalMakespan \leftarrow Makespan(S')$ 
  end
13   $t_{i+1} \leftarrow t_i \times 0.99$ 
14   $i \leftarrow i + 1$ 
end
15 Return  $FinalS$  as the good enough scheduling string
  found by the proposed algorithm and  $FinalMakespan$ 
  as its related total makespan.

```

dition. The above-mentioned procedure is summarized in Algorithm 2.

5 An Illustrative Example

In this section, a simple grid environment with only *two* resources which aims to service *four* grid jobs is presented to show how the proposed approach can be used. Afterwards, an inclusive analysis on the effect of internal configuration of a resource on the makespan of the resource is done.

5.1 A Simple Example

Assume a grid environment with *two* resources R_1 and R_2 . The configuration of the resources is shown in Table 2. The numbers used for input parameters of the SAN model shown in Table 2 are selected randomly, and can be replaced with any real value.

Table 2 Configuration of grid resources considered in the illustrative example

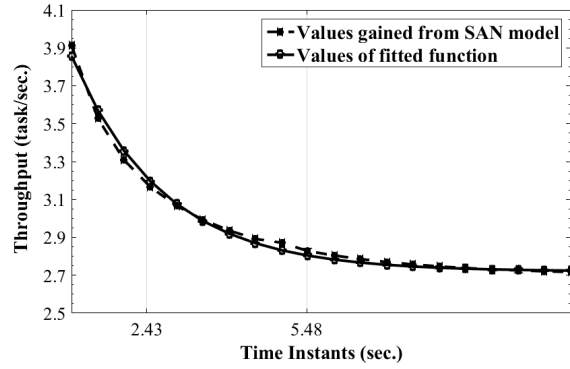
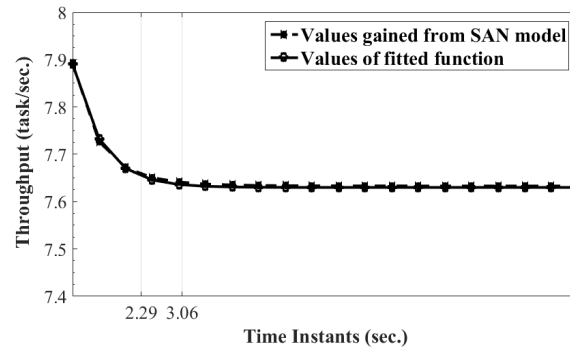
Parameters	R ₁	R ₂
Grid tasks arrival rate (λ_G) (tasks per second)	15.0	40.0
Local tasks arrival rate (λ_L) (tasks per second)	10.0	10.0
Processor service rate (μ) (tasks per second)	4.0	6.0
Grid queue size (M_G) (tasks)	15	20
Local queue size (M_L) (tasks)	5	20
Number of processors (N) (processors)	4	3
Failure rate of an idle processor (γ_i) (processors per second)	0.05	0.08
Failure rate of a busy processor (γ_b) (processors per second)	0.3	0.1
Repair rate of a processor (δ) (processors per second)	3.0	5.0

Moreover, suppose there are *four* grid jobs named $J_1, J_2, J_3,$ and J_4 with sizes 5, 10, 9, and 6 tasks, respectively, which should be scheduled on resources R_1 and R_2 . Let $S = 1212$ denote a possible scheduling string in this environment which assigns jobs J_1 and J_3 to resource R_1 , and jobs J_2 and J_4 to resource R_2 . After modeling both resources R_1 and R_2 using the proposed SAN, throughputs of them are computed using Möbius by applying Equation (1). The values gained from solving the corresponding SAN models of resources R_1 and R_2 together with the functions fitted to these values are presented in Fig. 3 and Fig. 4, respectively.

As can be seen in Fig. 3 and Fig. 4, functions found to approximate throughputs of resources R_1 and R_2 which are $2.72 + 2.03e^{(-0.58t)}$ and $7.63 + 1.69e^{(-1.87t)}$, respectively, can appropriately estimate these measures. Since the results reported in both Fig. 3 and Fig. 4 are very close to each other, we use the Normalized Root Mean Squared Error (NRMSE) as Equation (8) to show how the fitted functions can estimate the results gained from SAN models.

$$NRMSE = \frac{\frac{1}{M} \sum_{k=1}^M (MRT_{func}(k) - MRT_{SAN}(k))}{MRT_{SAN}^{max} - MRT_{SAN}^{min}} \quad (8)$$

where M is the number of observations which is equal to 20 in both Fig. 3 and Fig. 4, and MRT_{func} and MRT_{SAN} are the throughputs gained from the fitted

**Fig. 3** The throughput of resource R_1 for grid jobs gained from the proposed SAN and its fitted function**Fig. 4** The throughput of resource R_2 for grid jobs gained from the proposed SAN and its fitted function

function and the proposed SAN model, respectively. Moreover, MRT_{SAN}^{max} (MRT_{SAN}^{min}) shows the greatest (smallest) throughput gained from the SAN model amongst all $M = 20$ samples. Using Equation (8), the NRMSEs are computed as 0.023 and 0.014 for Fig. 3 and Fig. 4, respectively, which show very good fits between the observed and estimated data in both cases.

According to Equation (5) and the scheduling string considered for this example ($S = 1212$), we can find the time required for servicing job J_1 by resource R_1 , called t_1^{11} , as Equation (9) considering that the ready time of resource R_1 is 1 in the beginning ($t_0^{11} = 1$).

$$\int_{t_0^{11}=1}^{t_1^{11}} (2.72 + 2.03e^{-0.58t}) dt = 5 \quad (9)$$

Solving Equation (9) results in $t_1^{11} = 2.43$. Hence, the ready time of resource R_1 when it starts to service job J_3 should be updated to 2.43 ($t_0^{13} = t_1^{11} = 2.43$). Solving the corresponding equation for job J_3 , the time required for servicing job J_3 after job J_1 by resource R_1 is computed as $t_1^{13} = 5.48$. Therefore, the makespan of resource R_1 can be computed as $Makespan_1 = t_1^{13} = 5.48$.

Applying the same procedure to resource R_2 which intends to execute jobs J_2 and J_4 , the makespan of this resource is computed as $Makespan_2 = 3.06$. Using Equation (7), the total makespan of the assumed grid environment can be found as follows.

$$Makespan = \max(Makespan_1, Makespan_2) = 5.48 \quad (10)$$

Changing the scheduling string $S = 1212$, the measure $Makespan$ will change. Applying the proposed SA-based algorithm, we can find a suitable scheduling with the minimum $Makespan$ in this example, since the state space is limited. For example, the scheduling string $S' = 1221$ is one of the best possible schedules of jobs J_1 to J_4 on resources R_1 and R_2 with low threshold $l = 2$, which results in $Makespan = \max(4.42, 3.47) = 4.42$.

5.2 Sensitivity Analysis

In order to study the impact of internal configuration of a resource on its makespan, and the total makespan of the environment, we consider different scenarios in this subsection which combine different forms of failure-repair behavior of processors and execution priority between grid and local tasks. In the original SAN model shown in Fig. 2, it has been considered that there are both grid and local tasks' arrivals to a grid resource, and the resource services local tasks with higher priority compared to the grid tasks. Moreover, it has been assumed that processors inside a grid resource can fail when they are in both idle and busy states. Now, we consider *seven* further scenarios to study the impact of failure-repair behavior of processors and the execution priority of tasks on the makespan of a single resource, and consequently, the total makespan of the grid environment. In each scenario, we change the SAN model presented in Fig. 2 by easily adding (removing) some components to (from) the original SAN, or modifying the predicates/functions of input/output gates to satisfy the desired properties.

In scenarios 1 and 2, local tasks have higher priority over grid tasks, and in scenarios 5 and 6 grid tasks have higher priority against local ones. The same execution priority has been considered for both grid and local tasks in scenarios 3 and 4, and finally, a grid resource without local tasks has been considered in scenarios 7 and 8. In scenarios with *odd* numbers (scenarios 1, 3, 5, and 7), the failure-repair behavior of processors has been modeled, but in scenarios with *even* numbers (scenarios 2, 4, 6, and 8), this behavior has not been considered. As can be concluded, scenario 1 is the only

Table 3 The total makespans of the sample grid environment in *eight* different scenarios

Scenario	Makespan ₁	Makespan ₂	Makespan
1	5.48	3.06	5.48
2	3.29	2.99	3.29
3	2.82	2.63	2.82
4	2.60	2.59	2.60
5	2.12	1.91	2.12
6	2.04	1.89	2.04
7	2.11	1.91	2.11
8	2.04	1.89	2.04

case considered in Subsection 5.1. To be able to numerically compare different situations, we model both resources R_1 and R_2 with the configurations given in Table 2 in all *eight* scenarios. For the sake of brevity, we do not present all plots representing the throughputs of resources R_1 and R_2 , and their fitted functions here, but makespans of both resources and the total makespan of the environment are presented in Table 3.

As can be seen in Table 3, scenarios with *even* numbers (scenarios 2, 4, 6, and 8) which do not consider the failure-repair behavior of processors inside a resource show smaller makespans compared to their corresponding scenarios with *odd* numbers (scenarios 1, 3, 5, and 7), which model the failure-repair behavior of processors. It turns out that if we do not consider the failure property of processors in the model, the makespan of a resource for both grid and local tasks will decrease, since all processors will be fully reliable and they can service grid and local tasks without any failure. As a result, the makespan of a single resource for grid tasks, and consequently, the total makespan of the entire environment decreases in this situation.

Scenarios 3 and 4 show the case in which the same priority is considered for executing both grid and local tasks. As expected, the makespan of a resource for a grid task in this case is less than that of the resource when higher execution priority is assigned to local tasks against grid tasks (scenarios 1 and 2). Hence, for resource R_1 as an example, the makespan resulted in scenario 3 is 2.82, while this time in scenario 1 which assigns higher execution priority to local tasks over grid ones is 5.48. Similarly, if we compare the results reported for scenarios 2 and 4 which do not consider the failure-repair behavior of processors inside a resource, it can be seen that the makespan of resource R_1 in scenario 2 is 3.29, while it is 2.60 in scenario 4. Comparing scenarios 3 and 4 shows that the total makespan of the system decreases when the failure-repair behavior of processors is not taken into account, as mentioned

earlier. As shown in Table 3, the total makespan of the environment for grid tasks is 2.12 and 2.04 in scenarios 5 and 6, respectively. These values are less than values reported for scenarios 1 to 4, since we assign higher execution priority to grid tasks over local tasks in scenarios 5 and 6. Moreover, the total makespan reported for scenario 6 is less than the makespan reported for scenario 5, since we do not consider the failure-repair behavior of processors in scenario 6.

In scenarios 7 and 8, we assume that there is no local load on the resource, and all processors inside the resource are allocated to service grid tasks. As can be seen in Table 3, the makespans reported for scenarios 7 and 8 are very similar to the makespans of scenarios 5 and 6, respectively. Actually, this is a rational conclusion, since assigning higher execution priority to grid tasks over local tasks causes a local task to be executed when there is no waiting grid task in the resource. In other words, existence or nonexistence of a local task has no effect on executing a grid task. Therefore, the makespans reported for both resources R_1 and R_2 in scenarios 7 and 8 are the same as those reported for scenarios 5 and 6, respectively. There is a subtle point here which can only be seen in makespans reported for resource R_1 in scenarios 5 and 7. As can be seen in Table 3, the makespan of resource R_1 in situation in which there is a higher execution priority for grid tasks over local tasks is 2.12, and this measure for the situation in which there is no local task in the system is 2.11. Although the difference is very low and it cannot be seen in resource R_2 , sometimes it happens. This phenomenon can be justified according to the failure rate of idle and busy processors. Suppose in scenario 5, a local task enters the resource and finds a processor in an idle state (there is no waiting grid task in the system), so the processor is allocated to the local task, and its state is changed to busy processor. Since the failure rate of a busy processor is more than that of an idle processor, the processor servicing a local task fails faster than an idle processor. So, the probability that a newly arriving grid task finds an idle processor in an operational state is more than the probability that it finds a busy processor servicing a local task in an operational state, and then, preempts the servicing processor, which causes the makespan of resource R_1 in scenario 7 to be less than the makespan of R_1 in scenario 5. This is the reason behind the difference existing between total makespans of scenarios 5 and 7. It is worthwhile to mention that it rarely happens, and its effect on the total makespan is trivial.

In addition to analyze the makespan of a resource in various situations, we are also interested to study the mean response time ($E[R]$) of a resource for grid tasks

Table 4 The mean response time of resource R_1 for grid tasks ($E[R]$)

Cases	<i>first</i>	<i>second</i>	<i>third</i>	<i>fourth</i>
$E[R]$ (sec.)	3.51	2.09	0.83	0.82

Table 5 The failure probability of grid tasks inside resource R_1 (P_F)

Cases	<i>first</i>	<i>second</i>	<i>third</i>	<i>fourth</i>
P_F ($\times 10^{-2}$)	1.2	2.6	4.1	4.1

in different cases. To achieve this, we consider *four* different combinations of local and grid tasks' executions inside a resource, and study the effect of local tasks on evaluation of the steady state mean response time of a resource for grid tasks. In the *first* (*third*) case, we assign higher execution priority to local (grid) tasks over grid (local) tasks. In the *second* case, the same execution priority is assigned to both local and grid tasks, and in the *fourth* case, we assume that there is no local task in the system, and a resource only services grid tasks. In the *first* case, since it is preferred to execute local tasks even when there are grid tasks in the system, the mean response time of the resource for grid tasks will be a big number compared to the situation in which local and grid tasks have the same priority for execution (*second* case). If we assign higher execution priority to grid tasks against local tasks (*third* case), it is obvious that the mean response time of the resource for grid tasks will decrease compared to the situation considered in the *second* case. Finally, in the *fourth* case, all processing power of the resource is dedicated to grid tasks, so the mean response time of the resource for grid tasks will be the smallest one among all four cases considered. According to the explanation given about the makespans of a resource in scenarios 5 and 7, the mean response time of the resource for grid tasks in the *third* and *fourth* cases would be very close to each other, but it is expected for this measure to get its minimum value in the *fourth* case when there is no local task in the system. Table 4 shows the steady state mean response time of resource R_1 in four cases discussed above.

The failure probability (P_F) of grid tasks introduced in Subsection 4.2 is a measure which shows how many grid tasks fail during servicing inside a resource. It turns out that the probability P_F raises if the number of processors allocated to service grid tasks increases. Table 5 shows probability P_F for *four* different cases introduced above. As can be seen in Table 5, P_F gets its smallest value in *first* case amongst all other cases which shows that the number of processors allocated

to service grid tasks in *first* case is less than the others. Moreover, probability P_F gets its maximum value when grid tasks have higher execution priority over local tasks or there is no local load on the resource (*third* and *fourth* cases).

Now, having three measures *Throughput*, *Mean Response Time* and *Failure Probability* of a resource together, we can make a comprehensive analysis of behavior of the resource in executing grid tasks. When local tasks have higher execution priority over grid tasks, the *Throughput* of the resource for grid tasks is low (Scenarios 1 and 2 in Table 3), and the *Mean Response Time* of the resource for grid tasks is high (*first* case in Table 4). The *Failure Probability* of grid tasks in this situation is a small number (*first* case in Table 5). It shows that the number of processors allocated to execute grid tasks is kept as small as possible to service local tasks faster. The situation for grid tasks gets better, and more processors are allocated to service them if the same execution priority is assigned to both grid and local tasks (Scenarios 3 and 4 in Table 3, and *second* case in both Table 4 and Table 5), compared to the situation in which local tasks have higher execution priority over grid tasks. Finally, if we assign higher priority to execute grid tasks against local ones or we totally remove local tasks from the system, all processors inside a resource are allocated to service grid tasks even if there are waiting local tasks in the resource. The measure *Throughput* reported in Scenarios 5 to 8 in Table 3, and the measure *Mean Response Time* shown in *third* and *fourth* cases of Table 4 emphasize this fact that more processors are allocated to service grid tasks in these cases compared to two first cases discussed. Although increasing the *Failure Probability* of grid tasks which is done in *third* and *fourth* cases as shown in Table 5 is not a good event and perhaps not a good performance, but it is unavoidable because when the number of working processors increases, the probability of their failures also increases.

6 Case Studies

In order to apply the proposed approach to real grid environments, two comprehensive case studies are presented in this section. In *case study 1*, we investigate a real desktop grid called *lri* [21] with 40 resources ($m = 40$). The log of the system for one month can be downloaded from [1], but some internal specifications of the resources considered in the proposed SAN (e.g. grid and local queue sizes) have not been reported in the log. Since none of the workloads and logs reported from real grid systems contains all our required detailed information of the resources, we set some random numbers

Table 6 Configuration of grid resources considered in comprehensive case studies

Parameters	Ranges
Grid tasks arrival rate (λ_G)	[5, 20]
Local tasks arrival rate (λ_L)	[1, 5]
Processor service rate (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 (δ)	[5, 10]

to these parameters in the model regarding some research papers presented in this area [3, 8, 9, 10, 11, 13, 17, 22, 25, 26, 30, 31, 32, 36, 39, 40]. The configuration of the resources is shown in Table 6. As can be seen in this table, for each parameter is specified a range which shows that the value of the parameter is randomly selected in the specified range by Möbius tool.

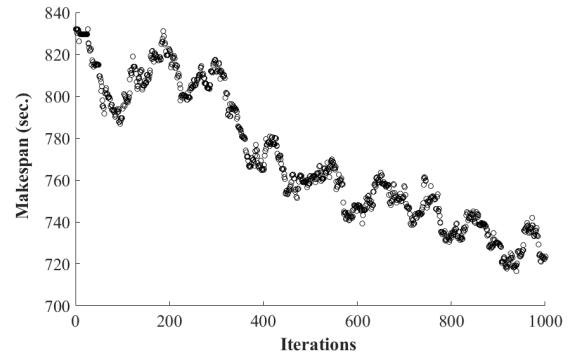
In *case study 1*, it is assumed that 500 independent jobs are submitted to the grid environment to be executed ($n = 500$). The jobs are independent and their sizes are randomly selected in range [1, 10]. For each resource, a SAN model is constructed and the throughput of the resource for grid jobs is computed. Afterwards, a function fitted to the results obtained by analyzing each SAN model is found, and the algorithm shown in Algorithm 1 is applied to each resource to find the makespan of the resource for a given scheduling string. Applying the SA-based scheduling algorithm presented in Algorithm 2, a scheduling string which results in a smaller makespan is achieved in each iteration. This procedure is continued until the termination condition is satisfied. In this case study, the number of iterations which is used as the termination condition is set to 1000. The makespans resulted from applying the proposed approach to *case study 1* are presented in Table 7 for different thresholds in range $l = 1$ to $l = \lfloor \frac{500}{40} \rfloor = 12$. It is worthwhile to mention that we have run the proposed algorithm 10 times for each value of parameter l , and then, got average from the results to produce more dependable results. The results reported in Table 7 are the averaged results. As can be concluded from Table 7, increasing the value of parameter l imposes the proposed algorithm to assign more jobs to each resource causing to assign more jobs to low-speed resources resulting in higher makespans. So, the makespan of the environment increases when the value of parameter l increases.

In *case study 2*, we consider a bigger grid environment with 200 resources ($m = 200$) and 2000 grid jobs

Table 7 The results obtained by applying the proposed approach to two comprehensive case studies

<i>case study 1</i>		<i>case study 2</i>	
Threshold l	Makespan	Threshold l	Makespan
1	692.86	1	2675.99
2	700.17	2	2686.88
3	707.96	3	2691.81
4	710.38	4	2698.45
5	719.73	5	2720.25
6	728.59	6	2738.94
7	735.96	7	2774.51
8	745.56	8	2782.37
9	749.70	9	2796.75
10	753.78	10	2813.08
11	755.03	-	-
12	760.93	-	-

($n = 2000$). Configurations of the resources considered in this case study are also taken from the ranges shown in Table 6 by Möbius tool. The makespans resulted by applying the proposed approach to *case study 2* are also presented in Table 7 for different thresholds in range $l = 1$ to $l = \lfloor \frac{2000}{200} \rfloor = 10$. As can be seen in Table 7, increasing the value of parameter l increases the resulted makespans as described for *case study 1*. It should be mentioned that the number of iterations in this case study is set to 200000, since the numbers of both resources and jobs are bigger than those of *case study 1*. Although choosing a small value for the number of iterations as the termination condition of the proposed SA-based scheduling algorithm decreases the time required for running the algorithm, the makespan resulted from few iterations will not be a precise result. On the other hand, if we set the number of iterations to a very large number, the execution time of the scheduling algorithm will increase and it will impose an overhead on the GM. Hence, it is a trade-off between the time required to execute an heuristic method, and the precision of the final result obtained from the method. This number can be easily found by checking some numbers and comparing the resulted makespans. If increasing the number of iterations does not affect the makespan so much, it shows that the algorithm is converging to the suitable result, and there is no need to increase the number of iterations further. For example, when the number of iterations in *case study 1* reaches 1000, the algorithm terminates and the resulting scheduling string together with its corresponding total makespan is reported as the output of the algorithm. If we set the iteration number to a bigger value (e.g. 2000 or 3000), the resulting value for the final outcome (the total makespan of the environment) will be approximately the same as that for 1000 itera-

**Fig. 5** The convergence of the proposed SA-based scheduling algorithm to a suitable result within 1000 iterations for $m = 40$, $n = 500$, and $l = 5$

tions. It shows that the proposed SA-based scheduling algorithm can converge to a suitable result in a timely manner, and the number 1000 is a good choice for this reason. Fig. 5 shows the convergence of the proposed SA-based scheduling algorithm to a result within 1000 iterations in *case study 1* for low threshold $l = 5$.

7 Conclusions and Future Work

The makespan of a grid environment is one of the most important Quality of Service (QoS) factors which should be studied in grids. Minimizing the makespan of a grid environment, the throughput of the environment increases. To minimize the total makespan of a grid, suitable job scheduling algorithms should be developed. A suitable job scheduling algorithm not only should consider the characteristics of an environment to be able to appropriately dispatch jobs to the resources, but also it should take into account the specification of jobs. For this reason, a SA-based scheduling algorithm is presented in this paper to dispatch grid jobs to grid resources with the aim of minimizing the total makespan of the grid environment. Before being able to use the proposed scheduling algorithm, each of the grid resources should be studied exactly to be able to assess their actual performance. To fulfill this requirement, a SAN model is presented to model and evaluate the performance of a single grid resource when the failure-repair behavior of its processors is taken into consideration. Using the proposed SAN model, we can model more detailed inner structure of a grid resource, and thereby, reach a more realistic estimation. In addition to the SAN model, two algorithms are presented in this paper to calculate the makespan of a single grid resource, and the total makespan of the whole grid environment based on the proposed SAN.

There are numbers of research issues remaining open for future work. The followings are some guidelines which can be used for further research in this area.

- *Considering different request classes for each of grid and local tasks and applying a mechanism to prioritize them inside a single queue.* This idea can be carried out by exploiting colored versions of SANs or PNs which let the modeler to distinguish tokens inside a place with different types. In this case, different mechanisms can be applied to service the tokens representing various types of grid/local tasks.
- *Taking into account the data and control dependencies between the jobs submitted to the environment.* In our proposed approach, none of the jobs has dependency to each other. All jobs are independent and each job can be serviced by a resource as soon as it is assigned to the resource. However, in practical systems, the jobs submitted to the system belong to a very big application, and therefore; there are some dependencies among the jobs of a single application. This practical issue can be considered in a future research as one of the most important extensions of the proposed approach.
- *Considering other structures of grid environments and various types of resource connections.* In this paper, it is assumed that all resources are independent and have no interaction with each other. This is a simple case of grid resource connection which has been assumed in many research papers to relax the problem to make it easier to solve by mathematical models, but generally, more complicated topologies can be considered for grid environments too.
- *Modeling the possibility of the GM detecting failures and reassigning jobs to different resources.* Using some intelligent methods, one can add the possibility of detecting failures to the GM which helps a grid environment to predict failures before happening and get the jobs from the failure-prone resource, and then, reassign it to another resource. In this mechanism, the new resource can execute the job from the point where it was stopped on the failure-prone resource.
- *Applying the proposed approach to the cloud computing environments and considering the specific characteristics of clouds in the proposed SAN and algorithms.* Since cloud systems are widely used nowadays, applying some modifications to the proposed approach can help us to be able to exploit it in analyzing cloud systems. Thinking about some important characteristics of clouds such as power consumption, cloud federation, virtual machine allocation, virtual machine migration and so forth can lead to new ideas in this research filed.

References

1. The Failure Trace Archive. <http://fta.scem.uws.edu.au/>. Accessed: February 2016
2. Abdual, W., Ramachandram, S.: Reliability-aware scheduling based on a novel simulated annealing in grid. In: The 4th International Conference on Computational Intelligence and Communication Networks, pp. 665–670. Phuket, Thailand (2012)
3. Azgomi, M.A., Entezari-Maleki, R.: Task scheduling modelling and reliability evaluation of grid services using coloured petri nets. *Future Generation Computer Systems* **26**(8), 1141–1150 (2010)
4. Azgomi, M.A., Movaghar, A.: A modeling tool for a new definition of stochastic activity networks. *Iranian Journal of Science and Technology Transactions* **29**(B1), 79–92 (2005)
5. Bolch, G., Greiner, S., Meer, H.D., Trivedi, K.S.: *Queueing Networks and Markov Chains*, second edn. John Wiley and Sons (2006)
6. Ciardo, G., Blakemore, A., Chimento Jr., P.F., Muppala, J.K., Trivedi, K.S.: Automated generation and analysis of markov reward models using stochastic reward nets. In: C.D. Meyer, R.J. Plemmons (eds.) *Linear Algebra, Markov Chains, and Queueing Models, The IMA Volumes in Mathematics and its Applications*, vol. 48, pp. 145–191. Springer (1993)
7. Daly, D., Deavours, D., Doyle, J.M., Webster, P.G., Sanders, W.H.: Mobius: An extensible tool for performance and dependability modeling. In: B.R. Haverkort, H.C. Bohnenkamp, C.U. Smith (eds.) *Computer Performance Evaluation: Modelling Techniques and Tools, Lecture Notes in Computer Science (LNCS)*, vol. 1786, pp. 332–336. Springer (2000)
8. Damodaran, P., Velez-Gallego, M.C.: A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Systems with Applications* **39**(1), 1451–1458 (2012)
9. Entezari-Maleki, R., Mohammadkhan, A., Yeom, H.Y., Movaghar, A.: Combined performance and availability analysis of distributed resources in grid computing. *The Journal of Supercomputing* **69**(2), 827–844 (2014)
10. Entezari-Maleki, R., Movaghar, A.: Availability modeling of grid computing environments using SANs. In: The 19th International Conference on Software, Telecommunications and Computer Networks, pp. 1–6. Split, Croatia (2011)
11. Entezari-Maleki, R., Movaghar, A.: A genetic algorithm to increase the throughput of the computational grids. *International Journal of Grid and Distributed Computing* **4**(2), 11–24 (2011)
12. Entezari-Maleki, R., Movaghar, A.: A probabilistic task scheduling method for grid environments. *Future Generation Computer Systems* **28**(3), 513–524 (2012)
13. Entezari-Maleki, R., Trivedi, K.S., Movaghar, A.: Performability evaluation of grid environments using stochastic reward nets. *IEEE Transactions on Dependable and Secure Computing* **12**(2), 204–216 (2015)
14. Foster, I., Kesselman, C.: *The Grid 2: Blueprint for a New Computing Infrastructure*, second edn. Morgan Kaufmann (2004)
15. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications* **15**(3), 200–222 (2001)

16. Garg, R., Singh, A.K.: Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology* **18**(2), 256–269 (2015)
17. Ghosh, R., Longo, F., Naik, V.K., Trivedi, K.S.: Modeling and performance analysis of large scale IaaS clouds. *Future Generation Computer Systems* **29**(5), 216–234 (2013)
18. Kaushik, A., Vidyarthi, D.P.: An energy-efficient reliable grid scheduling model using NSGA-II. *Engineering with Computers* **32**(3), 355–376 (2016)
19. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science, New Series* **220**(4598), 671–680 (1983)
20. Kolls, W.M.: *Curve Fitting for Programmable Calculators*, third edn. Imtec publisher (1984)
21. Kondo, D., Fedak, G., Cappello, F., Chien, A.A., Casanova, H.: Characterizing resource availability in enterprise desktop grids. *Future Generation Computer Systems* **23**(7), 888–903 (2007)
22. Koodziej, J., Khan, S.U., Wang, L., Kisiel-Dorohinicki, M., et al., S.A.M.: Security, energy, and performance-aware resource allocation mechanisms for computational grids. *Future Generation Computer Systems* **31**(1), 77–92 (2014)
23. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of grid resource management systems for distributed computing. *Journal of Software: Practice and Experience* **32**(2), 135–164 (2002)
24. Laarhoven, P.M.V., Aarts, E.H.L., Lenstra, J.K.: Job shop scheduling by simulated annealing. *Operations Research* **40**(1), 113–125 (1992)
25. Levitin, G., Dai, Y.S.: Service reliability and performance in grid system with star topology. *Reliability Engineering and System Safety* **92**(1), 40–46 (2007)
26. Longo, F., Ghosh, R., Naik, V.K., Trivedi, K.S.: A scalable availability model for Infrastructure-as-a-Service Cloud. In: *The 41st IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 335–346. Hong Kong (2011)
27. Meyer, J.F., Movaghar, A., Sanders, W.H.: Stochastic activity networks: Structure, behavior, and application. In: *The International Workshop on Timed Petri Nets*, pp. 106–115. Torino, Italy (1985)
28. Movaghar, A.: Stochastic activity networks: A new definition and some properties. *Scientia Iranica* **8**(4), 303–311 (2001)
29. Movaghar, A., Meyer, J.F.: Performability modeling with stochastic activity networks. In: *The 1984 Real-Time Systems Symposium*, pp. 215–224. Austin, TX, USA (1984)
30. Parsa, S., Entezari-Maleki, R.: RASA: A new grid task scheduling algorithm. *International Journal of Digital Content Technology and its Applications* **3**(4), 91–99 (2009)
31. Parsa, S., Entezari-Maleki, R.: A queuing network model for minimizing the total makespan of computational grids. *Computers and Electrical Engineering* **38**(4), 827–839 (2012)
32. Parsa, S., Entezari-Maleki, R.: Task dispatching approach to reduce the number of waiting tasks in grid environments. *The Journal of Supercomputing* **59**(1), 469–485 (2012)
33. Qureshi, M.B., Dehnavi, M.M., Min-Allah, N., Qureshi, M.S., et al., H.H.: Survey on grid resource allocation mechanisms. *Journal of Grid Computing* **12**(2), 399–441 (2014)
34. Rathore, N., Chana, I.: Load balancing and job migration techniques in grid: A survey of recent trends. *Wireless Personal Communications* **79**(3), 2089–2125 (2014)
35. Rathore, N., Chana, I.: Variable threshold-based hierarchical load balancing technique in grid. *Engineering with Computers* **31**(3), 597–615 (2015)
36. Reda, N.M., Tawfik, A., Marzok, M.A., Khamis, S.M.: Sort-mid tasks scheduling algorithm in grid computing. *Journal of Advanced Research* **6**(6), 987–993 (2015)
37. Saleh, A.I.: An efficient system-oriented grid scheduler based on a fuzzy matchmaking approach. *Engineering with Computers* **29**(2), 185–206 (2013)
38. Sanders, W.H., Meyer, J.F.: Stochastic activity networks: Formal definitions and concepts. In: E. Brinksma, H. Hermanns, J.P. Katoen (eds.) *Lectures on Formal Methods and Performance Analysis, Lecture Notes in Computer Science (LNCS)*, vol. 2090, pp. 315–343. Springer (2001)
39. Tao, Y., Jin, H., Wu, S., Shi, X., Shi, L.: Dependable grid workflow scheduling based on resource availability. *Journal of Grid Computing* **11**(1), 47–61 (2013)
40. Torabzadeh, E., Zandieh, M.: Cloud theory-based simulated annealing approach for scheduling in the two-stage assembly flowshop. *Advances in Engineering Software* **41**(10-11), 1238–1243 (2010)
41. YarKhan, A., Dongarra, J.J.: Experiments with scheduling using simulated annealing in a grid environment. In: M. Parashar (ed.) *Grid Computing, Lecture Notes in Computer Science (LNCS)*, vol. 2536, pp. 232–242. Springer (2002)