JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 7, No. 1, March 2013, pp. 44-52

An Efficient Scheduling Method for Grid Systems Based on a Hierarchical Stochastic Petri Net

Mohammad Shojafar*
Department of Information Engineering, Electronics and Telecommunications, Sapienza University of Rome, Rome, Italy
shojafar@diet.uniroma1.it

Zahra Pooranian
Department of Computer, Dezful Branch, Islamic Azad University, Dezful, Iran
zahra.pooranian@gmail.com

Jemal H. Abawajy
School of Information Technology, Deakin University, Melbourne, Victoria, Australia
jemal.abawajy@deakin.edu.au

Mohammad Reza Meybodi
Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran
mmeybodi@aut.ac.ir

Abstract
This paper addresses the problem of resource scheduling in a grid computing environment. One of the main goals of grid computing is to share system resources among geographically dispersed users, and schedule resource requests in an efficient manner. Grid computing resources are distributed, heterogeneous, dynamic, and autonomous, which makes resource scheduling a complex problem. This paper proposes a new approach to resource scheduling in grid computing environments, the hierarchical stochastic Petri net (HSPN). The HSPN optimizes grid resource sharing, by categorizing resource requests in three layers, where each layer has special functions for receiving subtasks from, and delivering data to, the layer above or below. We compare the HSPN performance with the Min-min and Max-min resource scheduling algorithms. Our results show that the HSPN performs better than Max-min, but slightly underperforms Min-min.

Category: Distributed computing

Keywords: Grid computing; Hierarchical stochastic Petri net (HSPN); Resource scheduling; Resource allocation; Modeling

I. INTRODUCTION

Recent trends in parallel processing system design and deployment have been toward the development of distributed network systems [1]. In particular, grid computing systems, in which networks of computers from multiple administrative domains are integrated to create large-scale virtual computers, are becoming more prevalent. Many dif-

Open Access http://dx.doi.org/10.5626/JCSE.2013.7.1.44 http://jcse.kiise.org

Received 27 January 2013, **Accepted** 28 February 2013
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

ferent groups have been involved in a collaborative effort to establish so-called *virtual organizations* (VOs). A VO may be established to perform a single task, and then disappear just as quickly [2, 3]. Grid computing has evolved beyond its roots in academic science, and is currently on the threshold of mainstream commercial adoption. One of the promises of grid computing is that it will enable applications to be executed across multiple sites. Grid computing technology is developing rapidly, and many have begun to run large-scale parallel applications on these systems [4]. Some of the applications require coordinated access to resources that are managed by autonomous entities. This coordinated access is known as *resource scheduling*.

In this paper, we address the problem of resource scheduling for global resource sharing in a grid computing environment. Resource scheduling is a crucial issue for the development of stable and effective production grid computing infrastructures [5]. However, since grid computing resources are distributed, heterogeneous, dynamic, and autonomous, deciding how to allocate the grid resources to meet the requirements of applications is a challenging and complex issue.

We propose a new approach to grid computing resource scheduling, the *hierarchical stochastic Petri net* (HSPN), to optimize grid resource sharing. Stochastic Petri nets are modeling tools that can easily be used to analyze and evaluate complex models of discrete event dynamic systems (DEDSs) for performance and reliability. Probabilistic models are automatically constructed, based on a set of results underlying the dynamic behaviors of these nets, from which the theory of untimed Petri nets is derived [6]. The HSPN categorizes resource requests in three layers, where each layer has special functions to receive subtasks from, and deliver data to, the layer above or below. This hierarchical model reduces the complexity of scheduling [7]. We compare the performance of the HSPN with the Min-min and Max-min resource scheduling algorithms, and our experiments show that the HSPN performs better than Max-min, but slightly underperforms Min-min.

The rest of this paper is organized as follows. Section II presents related work on grid computing resource scheduling, focusing on Petri nets. Section III outlines the grid resource scheduling problem, and Section IV discusses the HSPN, our proposed approach. Section V evaluates the performance of the HSPN, and compares it with the performance of Min-min and Max-min. Finally, Section VI presents our conclusions, including the advantages and disadvantages of the HSPN, and discusses future research directions.

II. RELATED WORK

Scheduling and resource allocation are important issues for grid computing. Considerable research over the last decade has addressed the problem of job scheduling for parallel and distributed systems. Abawajy and Dandamudi [7, 8] propose an online dynamic scheduling policy that manages multiple job streams, with the aim of improving the mean response time and system utilization. Yu and Buyya [9] present a general framework to facilitate directed acyclic graph (DAG) scheduling in grid systems. However, their research does not account for dynamic changes in grid computing resources. Several Petri net models, such as extended time Petri nets, colored Petri nets (CPNs), and stochastic Petri nets (SPNs), have been developed to address dynamic changes, and are considered to be effective tools for scheduling and resource allocation in grid computing systems [10-12].

Generally, Petri nets assign tasks to grid resources, using either a distributed scheme, or a hierarchical scheme [13, 14]. In a distributed scheme, a broker considers resources to be distributed states for each request, sends the request to the sites that contain the distributed resources, and receives results from the distributed sites' coordinators. In a hierarchical scheme, requests for resources from a site are arranged hierarchically. Resources are classified based on their geographical location in relation to the site making the request, and sites have coordinators at different levels. Tasks are sent through the hierarchical brokers, to sites that have the needed resources. A distributed scheme is better than a hierarchical scheme, when all tasks are locally requested on sites; a request is sent directly to the brokers on sites where the resource is available. Otherwise, a hierarchical scheme is better for resource allocation and responding to requests. In a hierarchical scheme, scheduling occurs in three layers, and each site has one broker and scheduler, to control and manage allocation of its resources.

A three-layer model based on a hierarchical time Petri net (HTPN) is presented in [10], with different Petri net models constructed for each level. The Petri net models that the HTPN uses for grid resource scheduling are different from the models in our proposed method. Also, the HTPN focuses on independent tasks, and does not consider dependent tasks. Dependent task scheduling is presented in [11], using an extended time Petri net. The grid resource scheduling model based on Petri nets is extended in [12], which includes a discussion of existing studies of grid scheduling using Petri nets, and proposes a four-level scheduling algorithm that considers independent tasks. In the present paper, we propose the HSPN, a hierarchical scheme that uses SPNs to schedule and allocate resources, based on the hierarchical and distributed schemes presented in [15]. Our preliminary version of this paper, which is published in [16, 23], introduced the algorithm, without considering the models in detail.

III. THE GRID RESOURCE SCHEDULING PROBLEM

The efficiency of our proposed algorithm for indepen-

dent task scheduling problem depends on minimizing makespan. When scheduling tasks in a grid network, the broker should map n tasks $T = T_1, T_2, \ldots, T_n$ to m available resources $R = R_1, R_2, \ldots, R_m$ in a way that minimizes the makespan, which is the maximum completion time for these tasks, as defined in Equation (1).

$$makespan = \max\left(\sum_{i=1}^{n} ready[j] + ETC[i,j]\right), \quad for,\ j = 1,2...m \tag{1}$$

where the *ETC* matrix gives each task's execution time for each resource. It is assumed that every task executes on some resource, and is not mapped to another machine, until its execution ends.

As an example, given two resources $\{R_1, R_2\}$ and eight tasks $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$, *ETC* might be as shown in Equation (2). The values show the runtime of each task on each resource, without considering the resource's queue. When we also take into account the fact that each resource has several tasks in its queue, we arrive at the makespan, as defined above. Each resource has a certain execution rate per one million instructions (MI) so it is better to separate tasks into resource groups based on their time and cost consumption. Each task includes a certain number of instructions, which is used in Equation (3) to calculate the task's execution time on each resource. The number of instructions for each task comes from a uniform distribution on [100 MI, 110 MI].

$$\begin{array}{c} \\ T1 \\ T2 \\ T3 \\ T4 \\ T5 \\ T6 \\ T7 \\ T8 \end{array} \begin{array}{c} R1\ \ R2 \\ \left[\begin{array}{cc} 2 & 1 \\ 1 & 3 \\ 3 & 4 \\ 4 & 6 \\ 5 & 1 \\ 3 & 3 \\ 1 & 4 \\ 2 & 5 \end{array}\right] \end{array} \tag{2}$$

For each (i,j):

$$ETC(i,j) = \frac{Instructions(T_i)}{resource\_excution\_per\_MI(R_j)}. \tag{3}$$

If we sort the tasks in the present example in order of their execution times, they will fall into the two groups shown in Table 1. In this table, each task's deadline is chosen at random.

Table 1. Tasks assigned to two separate groups, for allocation to the resources

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
R_1	▲3	▲1					▲2	▲4
R_2			▲3	▲4	▲1	▲2		

The queue for R_1 in Table 1 has T_2 as the first task and T_8 as the last task, while the queue for R_2 has T_5 as the first task and T_4 as the last task. Based on Equation (1), the makespan for these tasks is shown in the series of equations in (4).

$$\begin{aligned} &\text{Makespan } (T_1) = T_{2,1} + T_{7,1} + T_{1,1} = 1+1+2 = 4 \\ &\text{Makespan } (T_2) = T_{2,1} = 1 \\ &\text{Makespan } (T_3) = T_{5,2} + T_{6,2} + T_{3,2} = 1+3+4 = 8 \\ &\text{Makespan } (T_4) = T_{5,2} + T_{6,2} + T_{3,2} + T_{4,2} = 1+3+4+6 = 14 \\ &\text{Makespan } (T_5) = T_{2,2} = 1 \\ &\text{Makespan } (T_6) = T_{5,2} + T_{6,2} = 1+3 = 4 \\ &\text{Makespan } (T_7) = T_{2,1} + T_{7,1} = 1+1 = 2 \\ &\text{Makespan } (T_8) = T_{2,1} + T_{7,1} + T_{1,1} + T_{8,1} = 1+1+2+2 = 6 \end{aligned} \tag{4}$$

Providers will charge customers based on the time and cost of each task per second (makespan).

IV. OUR PROPOSED METHOD (HSPN)

In this section, we introduce the HSPN, our proposed three-level grid resource scheduling model.

Fig. 1 shows the high-level structure of the HSPN. Resources are connected through a three-level hierarchical network. The first level is a wide area network (WAN) that connects local area networks (LANs). The second level is composed of the LANs that connect computing resources (personal computers and high performance computers), and the third level connects storage resources and other resources. In this three-level scheduling scheme, unlike the hierarchical and distributed schemes, users submit all tasks to a home scheduler at their own site, rather than to a grid scheduler, which preserves the autonomy of grid resources, and makes it convenient for users to submit and supervise tasks. In addition, the three-level scheme adds a local scheduler between the home scheduler and the grid scheduler, which not only removes some pressure from the grid scheduler, but also makes it possible for tasks to be executed locally.

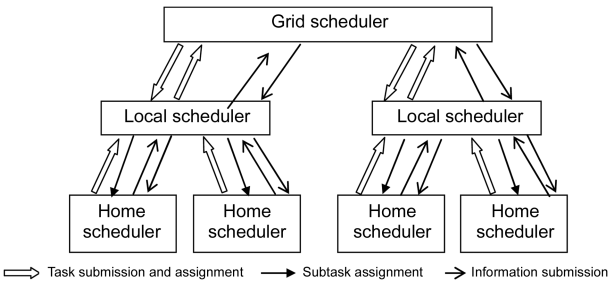

Fig. 1. The three layers for assigning tasks and subtasks.

Table 2. Notation for the hierarchical stochastic Petri net model

Place (P)	Description	Transition (T)	Description
P_1	Task submitted by user	t_0	Submitting task to home machine
P_2	Resources	t_1	Obtaining task and machine information
P_3	Scheduling task	t_2	Cannot complete task within deadline
P_4	Submitting task	t_3	Can complete task within deadline
P_5	Executing task	t_4	Sending task to local scheduler
P_6	Executing remote task	t_5	Assigning task to home scheduler
P_7	Task completed	t_6 or t_8	Executing remote or home task
P_8	Remote task	t_7	Submitting completed remote subtask
Pk_1,Pk_3,Pk_5	Output for submitting task Completing subtask Machine information	t_9	Returning completed task
Pk_2,Pk_4	Input for remote task Task completed	t_{10}	Providing machine information

Table 2 explains the notation for the HSPN model, showing the places and transitions that we use to simulate our hierarchical scheduling.

Each request for a grid resource is assessed and analyzed by the resource's local scheduler. Each task can be divided into several independent subtasks. Users submit tasks to their home scheduler with processing requirements, such as estimated processing time, estimated communication time, deadline, and degree of parallelism.

We have modeled these schedulers in the HSPN. The processes of task submission and assignment are as follows:

- A user submits a task to the home scheduler through the home machine. The home scheduler analyzes the submitted task. If the task can be completed within the deadline on the home machine, then the task will be executed on the home machine. Otherwise, the task is sent to the local scheduler.
- When the local scheduler receives the task submitted by the home scheduler, it decides whether the task can be completed within its deadline in the local area network. If so, the local scheduler assigns subtasks of the task to machines in the local area network, according to some algorithm. Otherwise, the task is sent to the grid scheduler.
- When the grid scheduler receives a task submitted by a local scheduler, it inserts the task into its queue of tasks.

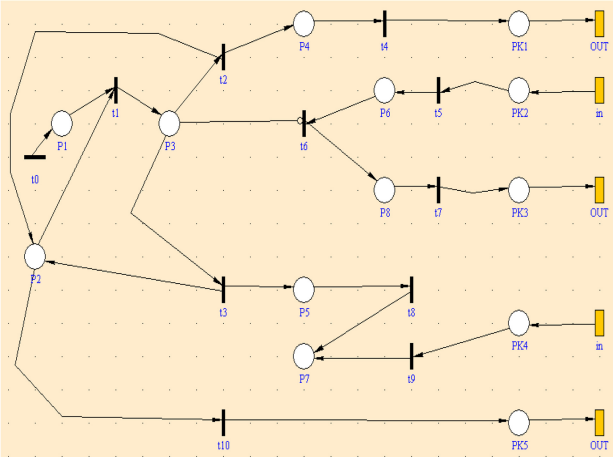

Fig. 2. Hierarchical stochastic Petri net model for resource scheduling.

The HSPN model, which depends on hierarchical scheduling, as shown in Fig. 1, is illustrated in Fig. 2. First, the home scheduler queue is searched. If the requested resource is not found on the home machine, then the request is sent to the local site coordinator, to try to find the best resource in the site, based on the QoS specified by the task parameters. The local scheduler searches the machines in its site. If the best resource is not found in that site, the coordinator sends a request to the site's broker, to contact brokers at other sites, to search for the resource in their sites. In this way, the grid scheduler enables a task to be allocated resources in other sites.

We implemented this model using GridSim [17], a resource scheduling simulator for grid systems. In GridSim, entities use events for both service requests, and service deliveries. An event can be raised by any entity for either immediate delivery, or delivery with a specified delay, to itself or other entities. Events that originate from the same entity to which they are delivered, are called *internal events*, and those that originate from external entities, are called *external events*. GridSim protocols are used

to define entity services. An event is called *synchronous,* when the event source entity waits until the event destination entity performs all the actions associated with the event (i.e., until the full service is delivered). An event is called *asynchronous,* when the event source entity that raises the event continues with other activities, without waiting for its completion. When the destination entity receives events or service requests, it responds by sending back one or more events, which can then take appropriate actions. External events can be synchronous or asynchronous, but internal events must be raised only as asynchronous events, to avoid deadlocks [17].

V. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our proposed approach, and compare it to the performance of two existing approaches.

A. Existing Scheduling Algorithms

We compare the HSPN with the Min-min and Max-min algorithms [18-21], briefly explained as follows.

1) Min-min computes the minimum completion time for each task over all machines. The task with the minimum of these completion times is selected, and assigned to the corresponding machine. The newly mapped task is removed from its waiting queue, and the process repeats, until all tasks are mapped.
2) Max-min is very similar to Min-min. The minimum completion time is calculated for each task, but the task with the maximum of these completion times is selected, and assigned to the corresponding machine.

We use identical uniform parameters for the Min-min and Max-min algorithms, when comparing them with the HSPN.

B. Testing Environment

We implemented the HSPN using GridSim package in NetBeans IDE 6.0 [22], each of which has four homogenous processors and a 1,000 GB storage unit. However, the machine at each site is different from the machines at the other sites, so that each site is heterogeneous with respect to the others.

There are 100 tasks that originate on each machine, and each task has an associated execution time, and execution cost. We used 100 tasks for each machine, because of the constraints of the underlying system environment; a greater number of tasks would not leave enough space in Java's heap for the grids. Table 3 shows the task states: there are four states, which are based on the resources that need to be run. Each task has a certain number of instructions, selected as a uniform distribution on [100 MI, 110 MI]. There are eight resources, and the resources each task needs to execute its instructions are defined by a uniform distribution on the range [1, 9], rounded up to an integer. Each resource has a queue of tasks that need that resource, and tasks are added to these queues, along with the number of instructions that comprise the tasks. So, when tasks are assigned, they are added to the resources that are needed. This means that each resource has a queue of tasks each which has an instruction number. All tasks are executed 100 times, to determine the best way to assign resources to tasks. The thinnest requests, for which it is easy to find resources, need only one or two resources, while the thickest requests need seven or eight.

Table 3. Task states

State	Max resources requests for (T_i)	Min resources requests for (T_i)
Thickest request	8	7
Medium request	6	5
Average request	4	3
Thinnest request	2	1

Table 4. Budget/time deadlines for the 100 tasks assumed in simulation and tests

Budget	Time deadline	Number of tasks
No	No	25-first
No	Yes	25-second
Yes	No	25-third
Yes	Yes	25-forth

The 100 tasks originating from each machine are separated into four groups, as shown in Table 4. The requests for each group of 25 are ordered, from the thinnest request to the thickest request. The four groups represent the four different states that tasks can be in, and are ordered based on their *ETC* values. This provides material for four tests in the HSPN, with 100 iterations to provide the best solutions.

The following section presents the results.

C. Performance Metrics

We used average response time and average response cost as metrics to evaluate the performance. These parameters are defined as follows:

- Average response time per request (ART): This metric evaluates the intervals between when a resource allocation request is received, and when the response

is sent. The response time in HSPN is calculated as the sum of the times needed to pass through the three layers, and depends on the request; we can say that ART is the makespan for each task.

- Average response cost per request (ARC): We define the response cost as the delivered data cost or resource cost for a task. All resource scheduling algorithms consider this metric. They usually consider the closest resources to the task site whose costs are less than the cost indicated for the task. However, it is possible for some of these resources to have costs that are slightly greater than the indicated cost. For example, if T_1 wants a resource that costs less than \$300, we search for resources that cost less than \$300 + ((1/100) × 300), or \$303. The ARC is the cost that the schedulers find for a response accepted by the user. In the HSPN, the ARC is the sum of the costs of passing through the three layers, and depends on the request. Some requests do not need to be sent to the grid layer or the local layer, so their cost is less than the cost for other methods.

D. Discussion of Results

Fig. 3 illustrates the ART per request coming from each user. The plot shows that Min-min is better than both Max-min and the HSPN for the first 25 tasks, with the HSPN falling between Min-min and Max-min. We considered the maximum time to be 36,000 seconds, and the maximum cost to be \$3,000. The times and costs increase from the 26th task to the 100th. As Fig. 3 shows, HSPN is near Min-min in ART, and it is better than Min-min for thick requests from the fourth class of the 25th task group. Moreover, the HSPN decreases the ART for Max-min by approximately 11%–12%. These results reflect the fact that some tasks found better resources, when searching the grid category.

Fig. 4 presents the average allotted time for each task for the three algorithms, and shows that the HSPN consumes an average of more than 150 units of time for the tasks, compared to Min-min; this is due to the context switches between layers. However, the HSPN consumes over 1,500 units less than Max-min, and hence decreases its average time allocation by approximately 10%.

Fig. 5 illustrates the ARC for each user's requests. The plot shows that for the first 25 tasks, which do not mention cost and time, all three algorithms deliver high value for each task; but for the second group of tasks, we considered a maximum cost of \$3,000. The scheduling should involve only the machines whose cost is less than the cost indicated for each task, which is Cost (Task) + ((1/100) ×

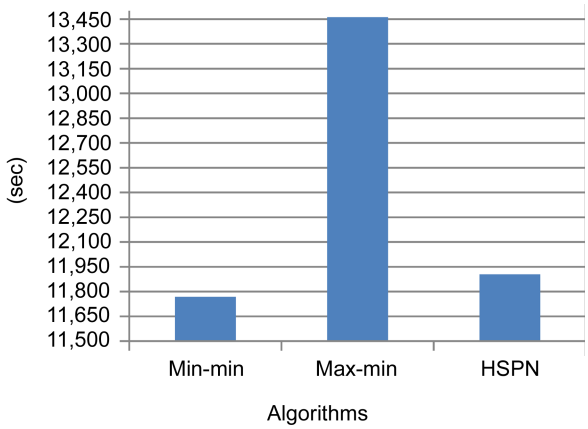

Fig. 4. Average time allocated for each task. HSPN: hierarchical stochastic Petri net.

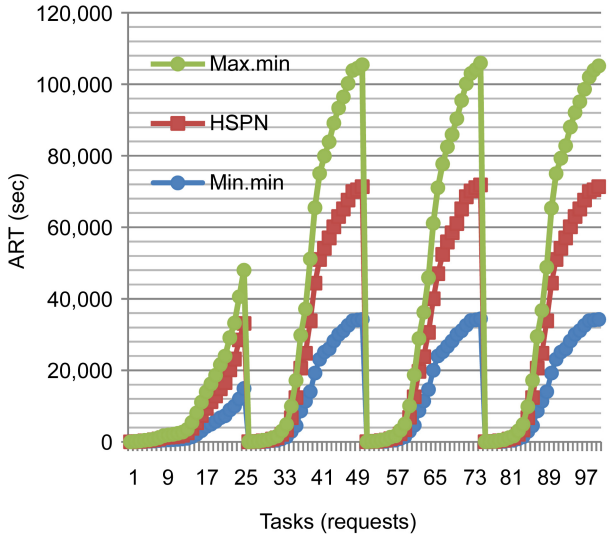

Fig. 3. Average response time (ART, or makespan) per request. HSPN: hierarchical stochastic Petri net.

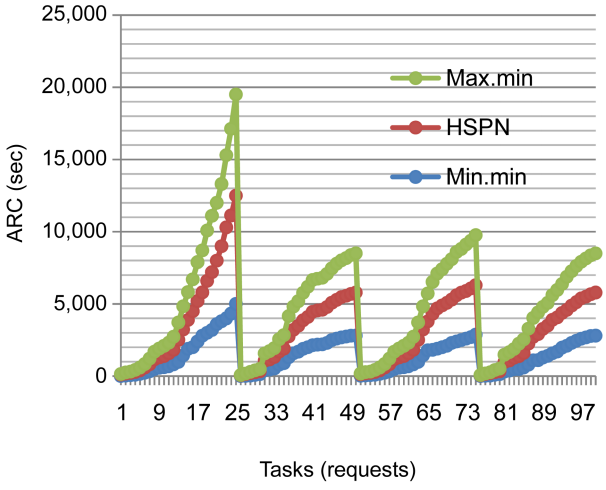

Fig. 5. Average response cost (ARC) per request. HSPN: hierarchical stochastic Petri net.

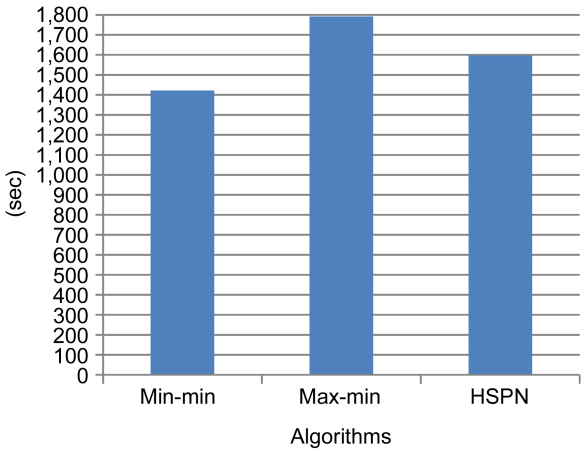

Fig. 6. Average cost for each task. HSPN: hierarchical stochastic Petri net.

Cost (Task)). As we can see, the HSPN decreases the cost by bounding the resources, but the decrease is not better than that obtained by Min-min. However, it is better than the cost obtained by Max-min. For thick requests, the performance of the HSPN is similar to Min-min, but sometimes it is better.

This is clearer in Fig. 6, which shows the average costs for tasks in the three algorithms. As can be seen, the average cost for the HSPN is about 170 units higher than for Min-min; this is related to the HSPN's context switch between layers. On the other hand, the HSPN decreases Max-min's average cost for 100 tasks by approximately 200 units, or approximately 11%.

VI. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

No method for scheduling and allocating resources is optimal for grid computing systems, because these systems are heterogeneous, dynamic, and expanding worldwide. This paper presented a three-level resource scheduling scheme for grid computing environments. The HSPN's resource scheduling scheme is modeled and analyzed by a hierarchical SPN, with different SPN models scheduling the resources of different layers (i.e., the home scheduler, the local scheduler and the grid scheduler).

We compared the HSPN with two major resource scheduling algorithms that are applicable in grid systems: Min-min and Max-min, and we used netbeans 6.0 IDE to analyze the results in a GridSim Jar file We used three queues for the schedulers, with Gridlet brokers sending requests between the layers. We evaluated the HSPN for 100 tasks, and found that it can decrease the execution time for tasks in on the order of 10%–11%, compared to Max-min. Moreover, the HSPN can be used for scheduling and allocating resources to tasks or subtasks. The HSPN can separate tasks based on their requested resources, but the other two methods cannot do so. Hence, the HSPN could see widespread use for scheduling in grid computing.

In the future, we plan to extend and implement our method for dependent tasks based on artificial intelligence algorithms, in order to decrease the scheduling time, and the time needed for resource allocation. We plan to improve the HSPN, to provide better results in comparison to the Min-min method.

REFERENCES

1. B. Javadi, J. H. Abawajy, M. K. Akbari, "Modelling and analysis of heterogeneous loosely-coupled distributed systems," School of Information Technology, Deakin University, Australia, Technical Report TR C06/1, 2006.
2. G. Koole and R. Righter, "Resource allocation in grid computing," *Journal of Scheduling*, vol. 11, no.3, pp. 163-173, 2008.
3. I. Foster, "The anatomy of the grid: enabling scalable virtual organizations," *Euro-Par 2001 Parallel Processing*, *Lecture Notes in Computer Science vol. 2150*, R. Sakellarius et al., editors, Heidelberg: Springer, pp. 1-4, 2001.
4. X. Wei, Z. Ding, S. Xing, Y. Yuan, and W. W. Li, "VJM: a novel grid resource co-scheduling model for parallel jobs," *International Journal of Grid and Distributed Computing*, vol. 2, no. , pp. 1-12, 2009.
5. J. H. Abawajy, "Adaptive hierarchical scheduling policy for enterprise grid computing systems," *Journal of Network and Computer Applications*, vol. 32, no. 3, pp. 770-779, 2009.
6. G. Balbo, "Introduction to stochastic Petri nets," *Lectures on Formal Methods and Performance Analysis*, *Lecture Notes in Computer Science vol. 2090*, E. Brinksma et al., editors, Heidelberg: Springer, pp. 84-155, 2001.
7. J. H. Abawajy and S. P. Dandamudi, "Scheduling parallel jobs with CPU and I/O resource requirements in cluster computing systems", in *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, Orlando, FL, 2003, pp. 336-343.
8. J. H. Abawajy and S. P. Dandamudi, "Parallel job scheduling on multicluster computing systems," in *Proceedings of the IEEE International Conference on Cluster Computing*, Hong Kong, China, 2003, pp. 11-18.
9. J. Yu, and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *Proceedings of the Workshop on Workflows in Surpport of Large-Scale Science*, Paris, 2006, pp. 1-10.
10. Y. Han, C. J. Jiang, and S. Luo, "Resource scheduling model for grid computing based on sharing synthesis of Petri net," in *Proceedings of the 9th International Conference on Computer Supported Cooperative Work in Design*, Coventry, UK, 2005, pp. 367-372.
11. Y. Han and X. Luo, "Modelling and performance analysis of grid task scheduling based on composition and reduction of Petri nets," in *Proceedings of the 5th International Confer-*

ence on Grid and Cooperative Computing, Changsha, China, 2006, pp. 331-334.
12. X. Zhao, B. Wang, and L. Xu, "Grid application scheduling model based on Petri net with changeable structure," in *Proceeding of 6th International Conference on Grid and Cooperative Computing*, Los Alamitos, CA, 2007, pp. 733-736.
13. Y. Han, C. Jiang, and X. Luo, "Resource scheduling scheme for grid computing and its Petri net model and analysis," *Parallel and Distributed Processing and Applications*, *Lecture Notes in Computer Science vol. 3759*, G. Chen et al., editors, Heidelberg: Springer, pp. 530-539, 2005.
14. Z. Hu, R. Hu, W. Gui, J. Chen, and S. Chen, "General scheduling framework in computational grid based on Petri net," *Journal of Central South University of Technology*, vol. 12, no. 1, pp. 232-237, 2005.
15. V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, "Distributed job scheduling on computational grids using multiple simultaneous requests," in *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing*, Edinburgh, UK, 2002, pp. 359-366.
16. M. Shojafar, S. Barzegar, and M. R. Meybodi, "A new method on resource scheduling in grid systems based on hierarchical stochastic Petri net," in *Proceedings of the 3rd International Conference on Computer and Electrical Engineering*, Chengdu, China, 2010, pp. 175-180.
17. R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175-1220, 2002.
18. B. Senthil Kumar, P. Chitra, and G. Prakash, "Robust task scheduling on heterogeneous computing systems using segmented MaxR-MinCT," *International Journal of Recent Trends in Engineering*, vol. 1, no. 2, pp. 63-65, 2009.
19. M. Y. Wu, W. W. Shu, and H. Zhang, "Segmented min-min: a static mapping algorithm for meta-tasks on heterogeneous computing systems," in *Proceedings of the 9th Heterogeneous Computing Workshop*, Cancun, Mexico, 2000, pp. 375-385.
20. T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, and M. Maheswaran, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing system," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810-837, 2001.
21. R. Armstrong, D. A. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Proceedings of the 7th IEEE Heterogeneous Computing Workshop*, Orlando, FL, 1998, pp. 79-87.
22. NetBeans IDE, http://netbeans.org/downloads/index.html?pagelang=.
23. M. Shojafar, S. Barzegar, and M. R. Maybodi, "Time optimizing in Economical Grid Using Adaptive Stochastic Petri Net Based on Learning Automata", in *Proceedings of International Conference on Grid Computing & Applications (GCA)*, Worldcomp, 2011, pp. 67-73.

Mohammad Shojafar

Mohammad Shojafar received his B.Sc. in Computer Engineering-Software major at Iran University Science and Technology, Tehran, Iran (2001-2006) and M.Sc. at Qazvin Islamic Azad University, Qazvin, Iran (2007-2010). He is Specialist in network programming in sensor field and Specialist in distributed and cluster computing (Grid Computing and P2P Computing), AI algorithms (Fuzzy Logic, Learning Automata, and Genetic algorithm), and Mathematical Optimization (Game Theory and Nonlinear programming). He published 6 papers in IEEE, 2 papers in WASET, and 2 papers in WORLDCOMP Conferences Series held in USA, and one ISI paper in IOS press till now. Mohammad was a Programmer and Analyzer in Exploration Directorate Section at N.I.O.C in Iran. He is a Ph.D. student at Sapienza University of Rome "La Sapienza" from 2012.

Zahra Pooranian

Zahra Pooranian received her M.Sc. in Computer Architecture degree as honor student in Dezful Islamic Azad University, 2011. She is an instructor in Sama University in Dezful and Ahvaz since 2009. Her research interest is in grid computing, specially in resource allocation and scheduling. She has worked on several papers in decreasing time and makespan in grid computing by using several AI methods such as GA, GELS, PSO, and ICA. She has published more than 5 papers, especially in grid scheduling and resource allocation in various conferences, such as WASET 2010-2011, ICCCIT 2011, and ICEEE 2011.

Jemal H. Abawajy

Jemal H. Abawajy is a faculty member in the School of Information Technology, Deakin University, Australia. He is actively involved in funded research in robust, secure and reliable resource management for pervasive computing (mobile, clusters, enterprise/data grids, web services) and networks (wireless and sensors) and has published more than 150 research articles in refereed international conferences and journals and books. He is currently the principal supervisor of 13 Ph.D. students and co-supervising 3 Ph.D. students. Prof. Abawajy is on the editorial board of several international journals. Prof. Abawajy has been a member of the organizing committee for over 100 international conferences serving in various capacities including chair, general co-chair, vice-chair, best paper award chair, publication chair, session chair and program committee.

Mohammad Reza Meybodi

Mohammad Reza Meybodi received the B.S. And M.S. degrees in Economics from Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received the M.S. and Ph.D. degrees from Oklahoma University, USA, in 1980 and 1983, respectively, in Computer Science. Currently, he is a full professor in Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to current position, he worked from 1983 to 1985 as an assistant professor at Western Michigan University, and from 1985 to 1991 as an associate professor at Ohio University, USA. His research interests include channel management in cellular networks, learning systems, parallel algorithms, soft computing and software development.