GLOA: A New Job Scheduling Algorithm for Grid Computing

[1]Zahra Pooranian, [2]Mohammad Shojafar, [3]Jemal H. Abawajy, and [4]Mukesh Singhal
[1]Graduate School, *Dezful Islamic Azad University, Dezful, Iran*
[2]Dept.of Information Engineering, Electronic and Telecommunication (DIET), *''Sapienza'' University of Rome, Rome, Italy*
[3]School of Information Technology, *Deakin University, Geelong, Australia*
[4]Computer Science & Engineering, *University of California, Merced, USA*

***Abstract* — The purpose of grid computing is to produce a virtual supercomputer by using free resources available through widespread networks such as the Internet. This resource distribution, changes in resource availability, and an unreliable communication infrastructure pose a major challenge for efficient resource allocation. Because of the geographical spread of resources and their distributed management, grid scheduling is considered to be a NP-complete problem. It has been shown that evolutionary algorithms offer good performance for grid scheduling. This article uses a new evaluation (distributed) algorithm inspired by the effect of leaders in social groups, the group leaders' optimization algorithm (GLOA), to solve the problem of scheduling independent tasks in a grid computing system. Simulation results comparing GLOA with several other evaluation algorithms show that GLOA produces shorter makespans.**

***Keywords* — Artificial Intelligence, Distributed Computing, Grid Computing, Job Scheduling, Makespan.**

I. INTRODUCTION

NEW technology has taken communication to the field of grid computing. This allows personal computers (PCs) to participate in a global network when they are idle, and it allows large systems to utilize unused resources. Like the human brain, modern computers usually use only a small fraction of their potential and are often inactive while waiting for incoming data. When all the hardware resources of inactive computers are collected as an all-in-one computer, a powerful system emerges.

With the help of the Internet, grid computing has provided the ability to use hardware resources that belong to other systems. "Grid computing" may have different meanings for different people, but as a simple definition, grid computing is a system that allows us to connect to network resources and services and create a large powerful system that has the ability to perform very complex operations that a single computer cannot accomplish. That is, from the perspective of the users of grid systems, these operations can only be performed through these systems. As large-scale infrastructures for parallel and distributed computing systems, grid systems enable the virtualization of a wide range of resources, despite their significant heterogeneity [1].

Grid computing has many advantages for administrators and developers. For example, grid computing systems can run programs that require a large amount of memory and can make information easier to access. Grid computing can help large organizations and corporations that have made an enormous investment to take advantage of their systems. Thus, grid computing has attracted the attention of industrial managers and investors in companies that have become involved in grid computing, such as IBM, HP, Intel, and Sun [2].

By focusing on resource sharing and coordination, managing capabilities, and attaining high efficiency, grid computing has become an important component of the computer industry. However, it is still in the developmental stage, and several issues and challenges remain to be addressed [3].

Of these issues and challenges, resource scheduling in computational grids has an important role in improving the efficiency. The grid environment is very dynamic, with the number of resources, their availability, CPU loads, and the amount of unused memory constantly changing. In addition, different tasks have different characteristics that require different schedules. For instance, some tasks require high processing speeds and may require a great deal of coordination between their processes. Finally, one of the most important distinctive requirements of grid scheduling compared with other scheduling (such as scheduling clusters) is scalability.

With more applications looking for faster performance, makespan is the most important measurement that scheduling algorithms attempt to optimize. Makespan is the resource consumption time between the beginning of the first task and the completion of the last task in a job. The algorithm presented in this paper seeks to optimize makespan. Given the complexity and magnitude of the problem space, grid job scheduling is an NP-complete problem. Therefore, deterministic methods are not suitable for solving this problem. Although several deterministic algorithms such as min-min and max-min [4] have been proposed for grid job scheduling, it has been shown that heuristic algorithms provide better solutions. These algorithms include particle swarm

DOI: 10.9781/ijimai.2013.218

optimization (PSO)[5], genetic algorithms (GAs)[6], simulating annealing (SA)[7], tabu search (TS)[8], gravitational emulation local search(GELS)[9], ant colony optimization (ACO) [10], and recently Learning Automata (LA) [26]. Also, some researchers have proposed combinations of these algorithms, such as GA-SA[11], GA-TS[12], PSO-SA[13], GPSO[14], and GGA[15].

It is important that an optimization algorithm for optimization problems should converge to the optimal solution in a short period of time. The group leaders optimization algorithm (GLOA) [16] was inspired by the influence of leaders in social groups. The idea behind the algorithm is that the problem space is divided into several smaller parts (several groups), and each part is searched separately and in parallel to increase the optimization speed. Each separate space can be searched by its leader, who tries to find a solution by checking whether it is the closest member to the local and global minimum.

In this paper, we use GLOA for independent task/job scheduling in grid computing. In addition to the simplicity of its implementation, GLOA reduces optimization time. The remainder of this paper is organized as follows. Section II discusses related methods. Section III presents a general model for job/task scheduling. Section IV presents the GLOA method and modifies it based on our problem. Section V compares simulation results obtained with this algorithm and several other heuristic algorithms. Finally, the last section presents the conclusion of this study.

II. Related Work

In [17], the TS algorithm, which is a local search algorithm, is used for scheduling tasks in a grid system. In [18], the SA algorithm is used to solve the workflow scheduling problem in a computational grid. Simulation results show that this algorithm is highly efficient in a grid environment. The TS algorithm uses a perturbation scheme for pair changing.

In [19], the PSO algorithm is used for job scheduling with two heuristic algorithms, latest finish time (LFT) and best performance resource (BPR), used to decide task priorities in resource queues. In [20], the critical path genetic algorithm (CPGA) and task duplication genetic algorithm (TDGA) are proposed; they modify the standard GA to improve its efficiency. They add two greedy algorithms to the GA so that the wait times for tasks to start and ultimately the makespan can be reduced. The proposed algorithms consider dependent tasks, so that computation costs among resources are considered as well. Chromosomes are divided into two parts, and the graph under consideration is transformed into a chromosome that performs mapping and scheduling. The mapping part determines the processors on which tasks will execute, and the scheduling part determines the sequence of tasks for execution. In the representation of a chromosome, task priorities are considered by examining the graph.

The CPGA algorithm combines the modified critical path (MCP) algorithm [21] and a GA. The MCP algorithm first determines critical paths, and if the parent of tasks being executed on a processor is executing on another processor, these tasks are transported to the parent's processor to reduce the cost of transportation between processors.

The TDGA algorithm combines the duplication scheduling heuristic (DSH) algorithm [22] and a GA. This algorithm first sorts tasks in descending order and then repeats the parent task on all processors so that the children can execute earlier, because the transportation cost between processors becomes zero. By repeating the parent task, overload and communication delays are reduced and total execution time is minimized.

The resource fault occurrence history (RFOH) [23] algorithm is used for job scheduling fault-tolerant tasks in a computational grid. This method stores resource fault occurrence histories in a fault occurrence history table (FOHT) in the grid information server. Each row of the FOHT table represents are source and includes two columns. One column shows the failure occurrence history for the resource and the other shows the number of tasks executing on the resource. The broker uses information in this table in the GA when it schedules tasks. This reduces the possibility of selecting resources with more occurrences of failures.

The chaos-genetic algorithm [24] is a GA for solving the problem of dependent task/job scheduling. This algorithm uses two parameters, time and cost, to evaluate quality of service (QOS), and chaos variables are used rather than randomly producing the initial population. This combination of the advantages of GAs and chaos variables to search the search space inhibits premature convergence of the algorithm and produces solutions more quickly, with a faster convergence.

The integer genetic algorithm (IGA) [25] is a genetic algorithm for solving dependent task/job scheduling that simultaneously considers three QOS parameters: time, cost, and reliability. Since these parameters conflict with one another and cannot be simultaneously optimized—as improvement of one reduces the quality of another—weights are assigned to each parameter, either by the user or randomly. If the user provides the weighting, the parameter that is more important to the user is given more weight than the others.

III. Problem Description

The problem studied in this paper is independent task/job scheduling in grid computing. The proposed algorithm should be efficient in finding a solution that produces the minimum makespan. Thus, the problem is to assign a set of m input tasks ($T=T_1,T_2,...,T_m$) to n resources ($R=R_1,R_2,...,R_n$), with the minimum makespan.

IV. The GLOA algorithm

GLOA is an evolutionary algorithm that is inspired by the effect of leaders in social groups. The problem space is divided into different groups, and each group has its own leader. The members of each group don't necessarily have similar characteristics, and they have quite random values. The

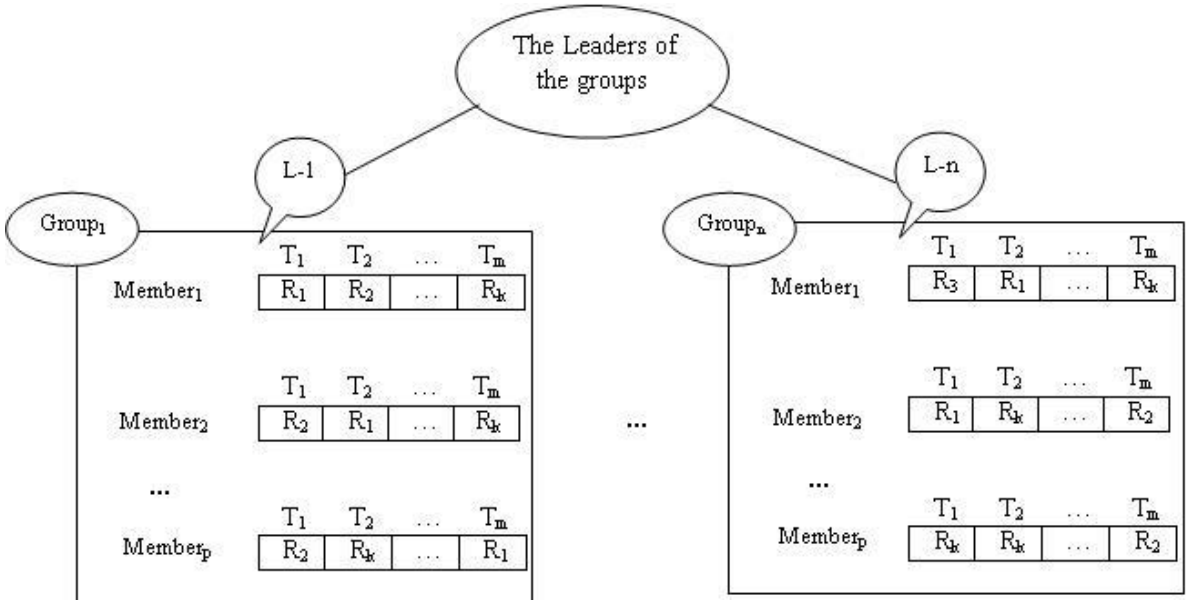

Fig. 1. Steps 1–3 of the algorithm: *n* groups consisting of *p* members are created, and their leaders are chosen based on their fitness values.

best member of each group is selected as the leader. The members of each group try to become similar to their leader in each iteration. In this way, the algorithm is able to search a solution space between a leader and its group members. It is obvious that after some iteration, members of a group may become similar to their leader. In order to introduce diversity within a group, one of its members is selected randomly and some of its variables are interchanged with a member of another group. In addition, a crossover operator helps a group come out of local minima, and the solution space can be searched again so to produce diversity. The algorithm steps are as follows:

A. Initial Population Production

A set of *p* members is produced for each group. The total population is therefore *n***p*, where *n* is the number of groups. Group and member values are produced randomly. Since the number of entering tasks is *m*, the members are represented as an *m*-dimensional array in which the stored values are resource numbers. For example, in Figure 1 we have *n* groups, each with *p* members.

B. Calculating Fitness Values of All Group Members

The fitness value is calculated for each member of each group. Since the purpose of task/job scheduling in a grid is to assign tasks to resources in a way that minimizes makespan, makespan has been chosen as the criterion for evaluating members. The less a member's makespan is, the greater is its fitness value, according to (1):

$$\text{fitness}(\text{member}_k) = \frac{1}{\text{makespan}(\text{member}_k)} \quad (1)$$

C. Determining Leader of Each Group

In each group, after the fitness value is computed for each member, the member with the best fitness value is selected as the group leader.

D. Mutation Operator

In this step, a new member is produced in each group from an older member, the leader of the group, and a random element, using (2). If the fitness value of the new member is better than the fitness value of the older member, it replaces the older member. Otherwise, the older member is retained.

$$\text{new} = r_1 * \text{old} + r_2 * \text{leader} + r_3 * \text{random} \quad (2)$$

where r_1, r_2, and r_3 are the rates determining the portion of the older member, the leader, and the random element that are used to generate the new population, such that $r_1 + r_2 + r_3 \leq 1$. Pseudocode for this step follows:

for i=1 **to***n* do {

for j=1 **to***p* do {

$\text{new}_{ij} = r_1 * \text{member}_{ij} + r_2 * Li + r_3 * \text{random}$

if fitness (new_{ij}) **better than** fitness (member_{ij})

then

$\text{member}_{ij} = \text{new}_{ij}$

end if

} end for

} end for

The value of r_1 determines the extent to which a member retains its original characteristics, and r_2 moves the member toward the leader of its group in different iterations, thus making the member similar to the leader. Careful selection of these two parameters plays an important role in the

optimization of the results. The main characteristic of this algorithm is that it searches the problem space surrounded by

TABLE I
PARAMETERS FOR THE ALGORITHMS

Algorithm	Parameter	Value
GLOA	Number of groups	3
	Population in each group	10
	r_1	0.8
	r_2	0.1
	r_3	0.1
GA	P-Crossover	0.85
	P-Mutation	0.02

the leaders. This leads to very rapid convergence to a global minimum. Note that eq. (2) is similar to the update equation for the PSO algorithm. The difference is that here, unlike PSO, the best position value of each member is not stored and so there is no information about the past positions of members.

TABLE II
THE ALGORITHMS' MAKESPAN AFTER 100 ITERATIONS (IN SECONDS)

(No. Tasks, No. Resources)	SA	GA	GSA	GGA	GLOA
(50,10)	136.742	99.198	95.562	90	89
(100,10)	307.738	183.49	190.353	181.028	167
(300,10)	973.728	638.082	626.66	597	581.842
(500,10)	1837.662	1105.56	1087	1087.216	1072.362

E. One-way Crossover Operator

In this step, a random number of members are selected from the first group and some of their parameter values are replaced with those of a member of another group that is selected randomly. It should be noted that in each iteration, only one parameter is replaced. If any new member is better it replaces the old one; otherwise the old member remains in the group. An important issue here is selecting the correct crossover rate, for otherwise all members will rapidly become similar to each other. The transfer rate t is a random number such that $1 \le t \le (\frac{m}{2}) + 1$ for each group. The purpose of the crossover operator is to escape local minima.

F. Repetition of Steps C to V according to the Determined Number of Iterations

This algorithm is repeated according to the determined number of iterations. At the end, from the different groups, the leader with the best fitness value is chosen as the problem solution.

V. SIMULATION

This section compares simulation results for our proposed algorithm with the results of several other algorithms. All algorithms were simulated in a Java environment on a system with a 2.66 GHZ CPU and 4GBRAM. Table I lists the parameters used in the performance study of our proposed algorithm and the other algorithms.

Table II shows the five algorithms' makespans for various numbers of independent tasks and 10 resources. As can be seen, SA has the worst makespans and GLOA has the best. We provide more details in Fig. 2.

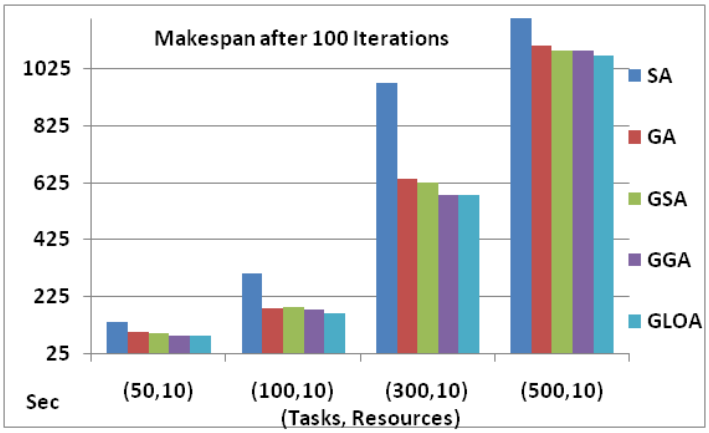

Fig. 2. Algorithms' makespan after100 iterations, with 10 resources

As we can see in Fig. 2, the SA algorithm's makespan increases rapidly as the number of tasks grows from 50 to 500.

TABLE III
ALGORITHMS' MAKESPAN AFTER 300 ITERATIONS (IN SECONDS)

(No. Tasks, No. Resources)	SA	GA	GSA	GGA	GLOA
(100,10)	233.2	172.628	179.062	175.598	166.14
(100,20)	173.116	111.946	105.314	103.092	94.55
(100,30)	120.452	90.716	87.846	80.086	77.75

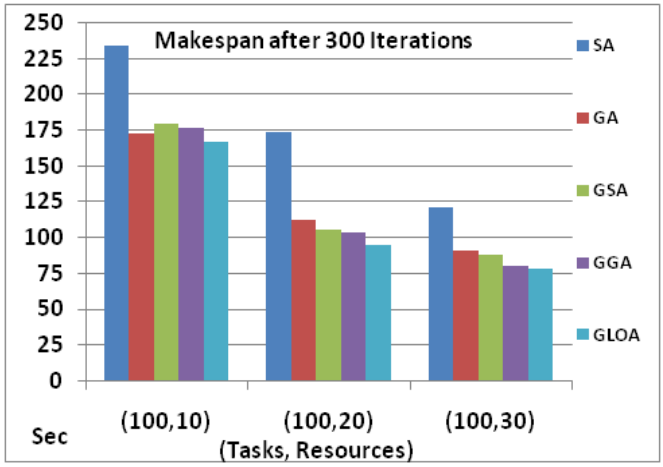

Fig. 3. Algorithms' makespan after 300 iterations, for various numbers of resources

Hence, SA is the worst algorithm for minimizing makespan and GLOA is the best in every case. In the 50-task case, the difference between SA and GLOA is approximately 48 seconds, which is less than half of the SA makespan. Here GLOA has the least makespan. When there are only a few tasks, the makespans for all of the algorithms are low, and GLOA produces the minimum. For the 300-task and 500-task cases, GGA has a similar makespan to the GLOA algorithm. For example, in the 300-task case, GGA's makespan is

approximately 597 seconds but GLOA's is approximately 582

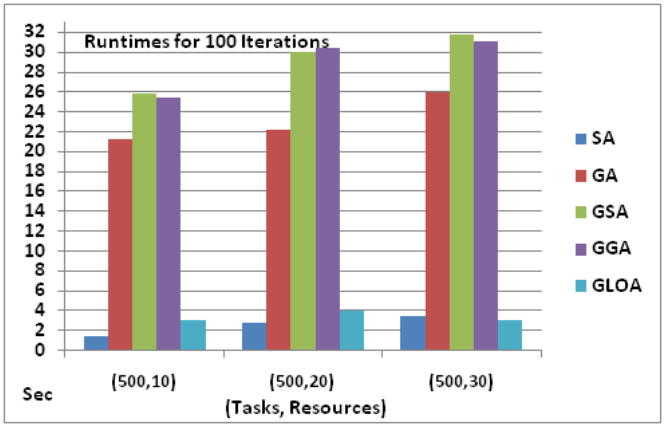

Fig. 4. Algorithm runtimes for 100 iterations with varying numbers of resources.

seconds.

Table III shows the makespans the algorithms produce for 100 fixed independent tasks for various numbers of resources. As can be seen, SA has the worst makespan in all of these cases and GLOA has the best. More details are shown in Fig. 3.

As can be seen in Fig. 3, as the number of resources increases, the makespan decreases for all algorithms, because when there are several ready resources with empty queues, tasks can be assigned to the new resources. The variation is the difference between the algorithms' structures. When the number of resources triples, the decrease for the makespan in SA is approximately 100 seconds and in GLOA it is approximately 90 seconds. As shown, GLOA has the minimum makespan in each case. Its structure provides it with the ability to be close to GGA, because like GGA, it can search the problem space both locally and globally. Hence, GLOA reaches the best solution more rapidly (e.g., in 95 seconds for 20 resources) than the other methods, particularly SA (which takes approximately 174 seconds). GLOA's makespan decreases up to 45% compared to SA, 15% compared to GA, 11% compared to GSA, and approximately 8% compared to GGA.

TABLE IV
ALGORITHMS' RUNTIME FOR 100 ITERATIONS (SECONDS)

(No. Tasks, No. Resources)	SA	GA	GSA	GGA	GLOA
(500,10)	1.4	21.2	25.8	25.4	3
(500,20)	2.8	22.2	30	30.4	4
(500,30)	3.4	26	31.8	31	3

Table IV shows the algorithms' runtime for job 500 independent tasks with varying numbers of resources. As shown, SA has the best runtime for 10 and 20 resources (because it considers only one solution, it can search more quickly than the other algorithms), and GLOA has the second best for 30 resources (because it divides the problem solutions into several groups that search in parallel, it reaches the optimum more quickly, but it takes some time to produce the several groups). Fig. 4 provides more details.

As can be seen in Fig. 4, when the number of resources increases, all algorithm runtimes increase, because when there are several new resources with empty queues, these resources must be searched and tasks assigned to them. SA is the least time-consuming algorithm (except for the 30-resource case) and GSA is the worst (except for the 500-task and 20-resource case). When the number of resources increases to 30, GLOA's runtime decreases less than SA's, because the resources have sufficiently many empty queues to be able to respond to 500 tasks more quickly, and SA considers the entire problem while GLOA divides the problem into several groups and considers the queue sizes and makespans for the tasks. When there are only a few resources (10), GA executes in just under 22 seconds, GSA and GGA have similar runtimes (just under 26 seconds), and GLOA requires just over 2 seconds, but SA requires less than 2 second. Although SA is the best algorithm in terms of runtime, it cannot produce better makespan results (as seen in Figure 2), and therefore we exempt this algorithm from consideration. When the number of resources triples (from 10 to 30), SA's runtime increases by 80%, GA's by 24%, GGA's and GSA's by 26%, but GLOA's increases by less than 10%. Therefore, while GLOA's runtime increases with the number of resources, it does so at a very low rate.

VI. CONCLUSION

Grid technology has made it possible to use idle resources as part of a single integrated system. The main purpose of grid computing is to make common resources such as computational power, bandwidth, and databases available to a central computer. The geographic spread and dynamic states of the grid space present challenges in resource management that necessitate an efficient scheduler. This scheduler should assign tasks to resources in such a way that they are executed in the shortest possible time.

This paper used a new evolutionary algorithm, GLOA, for scheduling tasks/jobs in a computational grid. Simulation results for GLOA were compared with results for four other intelligent algorithms: GA, SA, GGA, and GSA, and it was shown that in addition to wasting less computation time than the other algorithms, GLOA is able to produce shortest makespans. Also, GLOA could be applied in the real world because its runtime and makspan is less than other AI methods and produce less overhead on resources while responding the independent tasks.

In the future, we will change GLOA structure and apply it into dependent tasks in Grid Environment to cover the current gap into scheduling of dependent tasks.

REFERENCES

[1] J. Kołodziej, F. Xhafa, "Meeting security and user behavior requirements in Grid scheduling," *Simulation Modeling Practice and Theory, Elsevier*, pp. 213–226, 2011.

[2] S. Garg, R. Buyyaa, H. Siegel, "Time and cost trade-off management for scheduling parallel applications on Utility Grids," *Future Generation Computer Systems, Elsevier*, pp. 1344-1355, 2010.

[3] F. Xhafa, A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future Generation Computer Systems, Elsevier*, pp. 608-621, 2010.

[4] M. Shojafar, S. Barzegar, M. R, Meibody, "A new Method on Resource Scheduling in grid systems based on Hierarchical Stochastic Petri net," *3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)*, pp. 175-180, 2010.

[5] Q. Tao, H. Chang, Y. Yi, Ch. Gu, W. Li, "A rotary chaotic PSO algorithm for trustworthy scheduling of a grid workflow, " *Computers & Operations Research, Elsevier*, Vol. 38, pp.824–836, 2011.

[6] R. P. Prado, S. García-Galán, A. J. Yuste and J. E. M. Expósito, "Genetic fuzzy rule-based scheduling system for grid computing in virtual organizations," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, Vol. 15, No. 7, pp.1255-1271, 2011.

[7] S. Fidanova, "Simulated Annealing for Grid Scheduling Problem, International Symposium on Modern Computing," *IEEE John Vincent Atanasoff*, pp.41-45, 2006.

[8] B. Eksioglu, S. DuniEksioglu, P. Jain, "A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods," *Computers & Industrial Engineering*, 54, pp.1–11, 2008.

[9] B. Barzegar, A. M. Rahmani, K. Zamani far, "Advanced Reservation and Scheduling in Grid Computing Systems by Gravitational Emulation Local Search Algorithm," *American Journal of Scientific Research*, No. 18, pp. 62-70, 2011.

[10] Y. Yang, G. Wua, J.Chen, W. Dai, "Multi-objective optimization based on ant colony optimization in grid over optical burst switching networks," *Expert Systems with Applications*, 37, pp.1769–1775, 2010.

[11] G. Guo-ning , "Genetic simulated annealing algorithm for task scheduling based on cloud computing environment," *International Conference of Intelligent Computing and Integrated Systems (ICISS)*, pp-60-63, 2010.

[12] J.M. Garibaldi, D. Ouelhadj, "Fuzzy Grid Scheduling Using Tabu Search," IEEE International Conference of Fuzzy Systems, pp.1-6, 2007.

[13] R. Chen, D. Shiau, Sh. Tang Lo, Combined Discrete Particle Swarm Optimization and Simulated Annealing for Grid Computing Scheduling Problem, Lecture Notes in Computer Science, Springer, Vol. 5755, 2009, pp.242-251.

[14] Z. Pooranian, A. Harounabadi, M. Shojafar and J. Mirabedini, "Hybrid PSO for Independent Task scheduling in Grid Computing to Decrease Makespan," *International Conference on Future Information Technology (ICFIT 2011)*, Singapore, pp.435-439, 2011.

[15] Z. Pooranian, A. Harounabadi, M. Shojafar, N. hedayat, " New Hybrid Algorithm for Task Scheduling in Grid Computing to Decrease missed Task," *World Academy of Science, Engineering and Technology 79*, pp. 262-268, 2011.

[16] A. Daskin, S. Kais, "Group leaders optimization algorithm, Molecular Physics," *An International Journal at the Interface Between Chemistry and Physics*, Vol. 109, No. 5, pp. 761–772, 2011.

[17] M. Yusof, K. Badak, M. Stapa, "Achieving of Tabu Search Algorithm for Scheduling Technique in Grid Computing Using GridSim Simulation Tool: Multiple Jobs on Limited Resource," *International Journal of Grid and Distributed Computing*, Vol. 3, No. 4, pp. 19-32., 2010.

[18] R. Joshua Samuel Raj, V. Vasudevan, Beyond Simulated Annealing in Grid Scheduling, *International Journal on Computer Science and Engineering (IJCSE)*, Vol. 3, No. 3, pp. 1312- 1318, Mar. 2011.

[19] Ruey-Maw Chen and Chuin-Mu Wang, Project Scheduling Heuristics-Based Standard PSO for Task-Resource Assignment in Heterogeneous Grid, *Abstract and Applied Analysis*, Vol. 2011, pp.1-20, 2011.

[20] F. A. Omaraa, M.M. Arafa, "Genetic algorithms for task scheduling problem," *Journal Parallel Distributed Computing, Elsevier*, Vol. 70, No. 1, pp. 13-22, 2010.

[21] M. Wu, D. D. Gajski, "Hyper tool: A programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 3, pp. 330-343, 1990.

[22] H. El-Rewini, T. G. Lewis, H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*, Prentice-Hall, 1994, ISBN:0-13-099235-6.

[23] L. Khanli, M. Etminan Far , A. Ghaffari, "Reliable Job Scheduler using RFOH in Grid Computing," *Journal of Emerging Trends in Computing and Information Sciences*, Vol. 1, No. 1, pp. 43- 47, 2010.

[24] G. Gharoonifard, F. Moeindarbari, H. Deldari, A. Morvaridi, "Scheduling of scientific workflows using a chaos- genetic algorithm," Procedia Computer Science, Elsevier, Vol. 1, No.1, pp. 1445- 1454, 2010.

[25] Q. Tao, H. Chang, Y. Yi, CH. Gu, "A Grid Workflow Scheduling Optimization approach for e-Business Application," *Proceedings of the 10th International Conference on E-Business and E-Government*, pp. 168-171, 2010.

[26] J. A. Torkestani, "A New Distributed Job Scheduling Algorithm for Grid Systems," An International Journal of Cybernetics and Systems, Vol. 44, Issue 1, pp.77-93, 2013.

Zahra pooranian received her Msc in Computer Architecture degree as honor student in Dezful Islamic Azad University since 2011. She is an instructor in Sama University in Dezful and Ahvaz since 2009. Her research interest 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 2010-11, ICCCIT 2011, and ICEEE 2011.Author's profile

.

Mohammad Shojafar is PhD Student in Information Communication Technology at Sapienza University of Roma from November 2012. He Received his Msc in Software Engineering in Qazvin Islamic Azad University, Qazvin, Iran in 2010.Also, he Received His Bsc in Computer Engineering-Software major in Iran University Science and Technology, Tehran, Iran in 2006. Mohammad is Specialist in Network Programming in Sensor field and Specialist in Distributed and cluster computing (Grid Computing and P2P Computing) and AI algorithms (PSO, LA, GA).

Jemal H. Abawajy is Professor in School of Information Technology of Deakin University in Australia. His interest is building secure, efficient and reliable infrastructures for large-scale distributed systems. He has published 10 books, more than 55 international well-known Journals, more than120 Conference papers and 58 Book chapters in his interest. He supervised more than 31 PhD students till now. Also, he is teaching Grid Computing Security, Distributed Computing, Information Technology Security Management and Information Security Management in Deakin University now.

Mukesh Singhal is the Professor and Gartner Group Endowed Chair in Network Engineering, Dept. of Computer Science, The University of Kentucky, Lexington, USA; and Chancellor's Professor School of Engineering of University of California, Meced, USA. He has authored and published over 200 research papers and four textbooks which 175 of them are indexed in DBLP. The books on emerging topics that Dr. Singhal edited include, Distributed computing, Cloud computing, Security and Networks. He is Journal Editorships of 5 outstanding journals such as IEEE Trans. on Dependable and Secure Computing and IEEE Trans. on Parallel and Distributed Systems. He is one of the highly cited authors in computer science and software engineering worldwide(h-index = 39, Citations = 5594).