Dynamic and Adaptive Load Balancing using Harmony Search in Grid Computing

Fatemeh Heydari[†] and Hadi Shahriar Shahhoseini[††],

[†,††] Electrical Engineering Department, Iran University of Science and Technology, Tehran, Iran

Summary
Applications of network-based systems like cluster and grid systems have increased considerably in recent years. Load balancing is an important concept in these systems, which is implemented with the purpose of reducing execution time of applications and increasing utilization of resources. An algorithm is proposed in this paper for load balancing in the grid systems which operates on the basis of harmony search algorithm. This algorithm which is called Dynamic and Adaptive Harmony Search (DAHS) distributes the load among heterogeneous resources in a centralized and dynamic manner. DAHS parameters are defined adaptively in order to solve the difficulties of initialization. For creating a better load balancing, the amount of load in each resource is evaluated after formation of every harmony, such that a new task will not be assigned to it if a resource has extra load. The required simulations are done in GridSim simulator to investigate performance of this algorithm and it was demonstrated that utilization of resources was greater than 92% in DAHS method with mean square deviation being smaller than 0.06%. Meanwhile, makespan of the proposed method was found 10% lower in comparison with that of genetic method.

Key words:
Load balancing,Harmony search, Grid computing, Makespan

1. Introduction

Grid computing enables access to computing resources distributed in different positions for users [1],[2]. Such an environment has distinctive characteristics which distinguish them from other distributed systems and also conventional parallel processing systems. Heterogeneity, geographic distribution and being dynamic are some of the most important characteristics. Method of task scheduling and load balancing in these systems are of great significance due to their heterogeneous and dynamic features [1]-[3]. Load balancing includes distribution of tasks on the computing resources with minimum execution time and maximum utilization of resources.

One classification which has been presented for load balancing methods, involves three general schemes, namely centralized, distributed and hierarchical [4],[5]. In the centralized method, all tasks are assigned to one scheduler who is responsible for scheduling tasks on the resources. In this method, since all information related to scheduling are available, much better scheduling decisions would be made. In the distributed method, there is no central scheduler and scheduling is done by local schedulers. This method is developable though the produced scheduling might not be the optimal one. Distributed methods are divided into cooperative and non-cooperative methods based on whether they have cooperation with each other or not [6]. In this method all scheduling units are located at the same level, while different levels of management are defined in the hierarchical method. This method is in fact a combination of two previous methods. From another point of view they can be categorized as static, dynamic and adaptive [7]. In the static algorithms, decisions associated with load balancing are made within compile time and based on the information guessed. Advantage of static load balancing is its low overhead costs, because decision making is accomplished just once before calculations. However, this method can not conform to changes in computational requirements of applications and state of system. On the other hand, dynamic algorithms react toward changes in state of the system and allocate tasks to resources during execution time. In dynamic load balancing, it is possible to reassign tasks to processors upon running the program. If a node has extra load in the system, the task which has caused the extra load shall be transferred to another node and processed there. Task migration might incur great overhead costs to the system. Dynamic load balancing strategies are usually preferred to static load balancing strategies when this overhead cost is controlled reasonably. Adaptive algorithms are a special type of dynamic algorithms. They accommodate their activities with changes in state of the system which is done through changing characteristics of the algorithm [8].

Numerous load balancing algorithms are provided in this field so far some examples of which will be discussed here. Zomaya and Teh introduced a dynamic genetic algorithm for load balancing in distributed systems. They assumed that the characteristics of tasks can be defined in advance [9]. Le et al. presented algorithm which was a combination of FCFS and genetic algorithms [10]. In this algorithm, since there are small number of tasks in the queue, FCFS algorithm can be used alternatively. The purpose of this

Manuscript received September 5, 2012
Manuscript revised September 20, 2012

algorithm is to reduce execution time of the selected task considering the information existing in the central scheduler. Once the number of tasks available in the queue is increased, genetic algorithm is used for load balancing. In this situation, objective function is comprised of makespan and mean square deviation of utilization from resources. One major disadvantage of these two algorithms is their not being adaptive. Ludwig et al. have presented two distributed evolutionary algorithms for load balancing, one based on ant colony optimization (ACO) and the other based on particle swarm optimization (PSO) [11]. Performance of these algorithms has been compared in terms of makespan and load balancing level with stochastic algorithms and other common algorithms. The obtained results indicate that the proposed PSO algorithm has a superior performance as compared with the proposed ACO algorithm. Maximum level of load balancing in these algorithms is 81%.

Authors of this paper in [12] have used dependent task scheduling in distributed systems statically which leads to improved results of the suggested algorithm versus other harmony algorithms.

Harmony search (HS) has been utilized for load balancing in the grid, which distributes independent and heterogeneous tasks in a dynamic and adaptive manner on heterogeneous resources. Due to adaptive nature of the algorithm, problems of choosing appropriate parameters for it are solved. Task migration from overloaded resources to underloaded resources is one step of dynamic load balancing algorithms. Since task migration is rather costly and practically complicated during execution or after being allocated to the resources, it is usually overlooked in some load balancing algorithms [13],[14]. One load rebalancing is implemented on it to solve this problem after generating a new solution. In this case, if a resource has some extra load then the new tasks which are assigned to it will be handed over to the resource having load shortage. Moreover, to evaluate fitness of each solution, three criteria are considered: makespan, average utilization of resources and mean square deviation of resources load.

The remaining paper is organized as follows. In sections 2 and 3 load balancing and harmony algorithms are briefly introduced. The proposed algorithm will be presented in section 4, while simulation results have been demonstrated in section 5. Finally, conclusions and future works will be explained in section 6.

2. Load Balancing

One important issue in grid systems is load balancing which is implemented in order to reduce execution time and enhance performance. The system adopted in this work has been shown in Fig. 1. This system includes N heterogeneous resource with different processing power and bandwidth. Processors are connected to each other thoroughly. It has been assumed that the processors are connected to each other without any interference. Tasks are independent and indivisible. Tasks can be executed on each of the resources having stochastic entry time and Poisson distribution. Grid information service (GIS) is a unit which collects dynamic information of resources in definite time intervals and offers them to the scheduler.

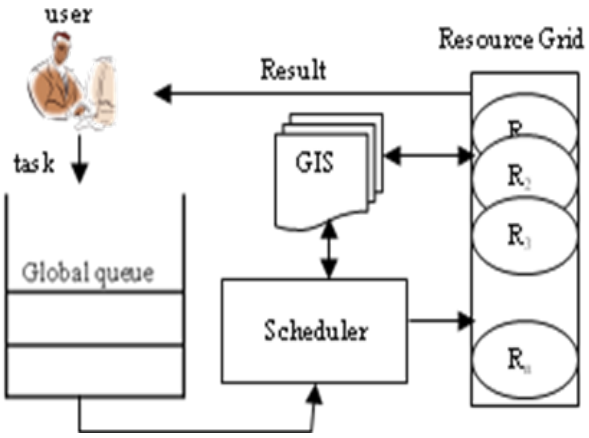

Fig.1 system model

After sending the tasks by the user, they are placed in a central queue which is related to the scheduler. A request is sent for the scheduler when one of the resources becomes idle. The scheduler selects from existing tasks in the queue according to the size of sliding window and executes the load balancing algorithm. If the number of tasks waiting in the queue is smaller than the sliding window, the scheduler will wait for a specific time and if the number of tasks still remains smaller than the size of window, it will select the same number of tasks and transfers them for the resources. Harmony algorithm is executed after selecting the tasks and then the tasks are sent to the appropriate resource.

3. Harmony Search Algorithm

Harmony search algorithm is an evolutionary algorithm introduced by Geem et al. in 2001 for the first time [15]. This algorithm imitates musical improvisation process. Every solution is called a harmony in this algorithm. At first, population of harmonies are generated stochastically and then stored in harmony memory (HM).

HS generates the new solutions using three principles, namely harmony memory consideration, pitch adjusting and random selection [15-17]. Assuming the new harmony

as $H_{new} = \{h_{new}(1), h_{new}(2), ..., h_{new}(N)\}$, this process can be implemented as below:

$$h_{new}(j) = h_i(j)\ i \in (1,2,..., HMS), \quad (1)$$

$$h_{new}(j) = h_{new}(j) \pm rand() \times bw, \quad (2)$$

$$h_{new}(j) = h_L(j) + rand() \times (h_U(j) - h_L(j)), \quad (3)$$

where, $rand()$ is a random number with uniform distribution within range of [0,1]. $HMCR$, PAR and HMS represent harmony memory consideration rate, pitch adjusting rate and harmony memory size, respectively. $h_L(j)$ and $h_U(j)$ are lower limit and upper limit of the variable under study, respectively, while bw stands for desired bandwidth.

Having generated the new harmony, if its fitness value outperforms that of the worst harmony in memory, it will be replaced and harmony generation is repeated until termination condition is met.

4. Proposed Algorithm

4.1 Coding

In DAHS, a harmony is an array whose length is equal to the size of sliding window. Each harmony represents the resources to which the tasks are allocated. Fig. 2 shows the harmony for load balancing problem with six tasks.

J_1	J_2	J_3	J_4	J_5	J_6
3	1	1	2	3	1

Fig. 2 Harmony representation

4.2 Fitness Function

Fitness function is a key feature of load balancing algorithm which in fact indicates the amount of fitness for each solution. Effective criterions of the fitness function are introduced here first and then the fitness function is expressed.

Eq.(4) describes how to calculate execution time of the tasks allocated to each resource which is called load of each resource hereafter. This value is comprised of the time needed to accomplish calculations and the time consumed to transfer tasks to resources.

$$load(R_i) = loadcurrent(R_i) + \sum_{j=1}^{m} \frac{tasklengh(j)}{pp(R_i)} + \sum_{j=1}^{m} \frac{outputsize(j) + inputsize(j)}{BW(R_i)}, \quad (4)$$

where, R_i is the desired resource, $load\ current$ is the current load of each resource, $tasklengh$ gives the processing requirement of each task, $pp(R_i)$ represents the processing power of each resource, BW is the bandwidth between each resource with scheduler, and finally, $outputsize$ and $inputsize$ offer the sizes of output and input files, respectively. Makespan is the completion time of the last task allocated to resources which is provided by Eq.(5). Meanwhile, N is the number of resources.

$$Makespan = \max(load(R_i))\ i = \{1,2,..., N\} \quad (5)$$

Utilization of each resource is defined as the ratio of desired resource load to makespan which is explained by Eq.(6). Average utilization of each resource is stated in Eq.(7) as the criterion effective on fitness of a solution.

$$U(R_i) = \frac{load(R_i)}{Makespan}. \quad (6)$$

$$AveU = \frac{\sum_{k=1}^{N} U(k)}{N} \quad (7)$$

The next criterion which is used to assess fitness of a solution is mean square deviation of load (Eq.(8)).

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (load_i - \overline{load})^2}{N}}. \quad (8)$$

Fitness function is defined in this algorithm as a function of makespan, average utilization of resources and mean square deviation of load, which is explained by Eq.(9). The fitness function is inversely related to makespan and mean square deviation. It is expected to decrease by increasing each criteria of solution fitness. Fitness function is directly related to the average utilization of resources because increased average utilization is indicative of an acceptable load balancing. Each criterion is applied with a specific coefficient in the fitness function which can be modified

according to important of each for the user. Simulations have been launched here assuming that α=β=γ=1.

$$fitness = \frac{\alpha \times AveU}{\beta \times Makespan \times \gamma \times \sigma} . \quad (9)$$

4.3 Algorithm Parameters

Previous studies reveal that definition of $HMCR$, PAR and bw in an adaptive manner improves performance of HS [12],[17].

In this paper, $HMCR$, PAR and bw in are considered as functions of harmony fitness generated in the earlier iteration which are described by Eq.(10), Eq.(11) and Eq.(12), respectively. Parameters are the same as the ones used in [12]. $HMCR$ will be increased if the harmony of former iteration shows a worse fitness versus the best harmony available. Thereby, the existing harmonies in the memory are used at a higher probability and changes are less probable to be introduced on them. Otherwise, $HMCR$ will be decreased to choose the harmony having greater probability stochastically. This will cause not to be located in an inappropriate position so that search will become possible throughout the whole solution space.

$$HMCR(k) = \begin{cases} HMCR_{\min} + (HMCR_{\max} \\ - HMCR_{\min}) \times \frac{f_{best}}{f_{k-1}} & f_{k-1} \geq f_{best} \\ HNCR_{\max} & f_{k-1} < f_{best} \end{cases} \quad (10)$$

$$PAR(k) = \begin{cases} PAR_{\max} - (PAR_{\max} \\ - PAR_{\min}) \times \frac{f_{best}}{f_{k-1}} & f_{k-1} \geq f_{best} \\ PAR_{\min} & f_{k-1} < f_{best} \end{cases} \quad (11)$$

$$bw(k) = \begin{cases} bw_{\max} - (bw_{\max} \\ - bw_{\min}) \times \frac{f_{best}}{f_{k-1}} & f_{k-1} \geq f_{best} \\ bw_{\min} & f_{k-1} < f_{best} \end{cases} \quad (12)$$

4.4 Load Rebalancing

When the resource has extra load and a task has been allocated to it, then the task will be transferred to the resource with the lowest load.

4.5 Process of DAHS Algorithm

Different steps of this algorithm are listed below:

Step 1: Parameters of the algorithm such as HMS, PAR, $HMCR$ and bw must be initialized first.

Step 2: Harmony memory is generated stochastically by uniform distribution.

Step 3: Algorithm parameters are defined adaptively.

Step 4: A new harmony is created.

Step 5: Load rebalancing is implemented.

Step 6: If the fitness of the new harmony outperforms that of the fitness of worst harmony available, then this harmony will be replaced by the worst harmony and the algorithm will return to step three in order to realize one of the two termination conditions, namely definite number of iterations and constant fitness of the best harmony available for 10% of total number of iterations.

5. Simulation Results

A grid environment including a number of users and ten resources has been taken into account for the purpose of simulation. The resource characteristics of which are summarized in Table 1. The resources defined are testbed resources of WWG extracted from literature [18]-[20]. The used simulation toolkit is GridSim.

Table 1: Characteristics of resources [20]

Resource name	Simulated resource characteristics Vendor, Resource type, Node OS, No of PEs	APE SPEC/MIPS Rating
R_0	Compaq, AlphaServer, CPU, OSF1, 4	515
R_1	Sun, Ultra, Solaris, 4	377
R_2	Sun, Ultra, Solaris, 4	377
R_3	Sun, Ultra, Solaris, 4	377
R_4	Intel, Pentium/VC820, Linux, 2	380
R_5	SGI, Origin 3200, IRIX, 6	410
R_6	SGI, Origin 3200, IRIX, 16	410
R_7	SGI, Origin 3200, IRIX, 6	410
R_8	Intel, Pentium/VC820, Linux, 2	380
R_9	SGI, Origin 3200, IRIX, 4	410
R_{10}	Sun, Ultra, Solaris, 8	377

In the following, the results of this method will be compared with genetic algorithm to investigate performance of DAHS. The genetic algorithm provided in [9] is a dynamic load balancing algorithm which is chosen as an appropriate reference for the purpose of comparison [10],[21]. Parameters of this algorithm have been listed in Table 2. The genetic algorithm under study is abbreviated as DGA hereafter.

Table 2: Parameters of DGA

Parameters	values
Window size	10
Crossover rate	0.9
Mutation rate	0.14
Population size	20
Number of generation	20

The size of harmony memory is defined as 5 for DAHS, while number of iterations and sliding window size have been assumed as 100 and 10, respectively.

5.1 Determination of Harmony Memory Size

Harmony memory size is one of the effective factors on algorithm accuracy. A large harmony memory would increase divergence which enhances the probability to choose the worse element. However, a very small harmony memory could increase the possibility to be located within local optima. Therefore, selection of a reasonable size of the harmony memory can establish equilibrium between divergence and convergence of the solutions.
In order to achieve an appropriate value for harmony memory size with a system whose characteristics are summarized in Table 1, diagram of makespan in terms of harmony memory size is are illustrated in Fig. 3, for 300 tasks having a length of 4000 to 8000, respectively. To mitigate the stochastic effect of results, this experiment was repeated 50 times. Results of simulation show that the harmony memory size of 5 offers the minimum makespan. Furthermore, this diagram confirms reduced performance of very large and very small harmony memories.

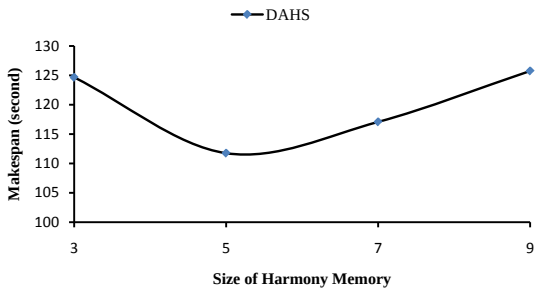

Fig. 3 Makespan of DAHS algorithm versus harmony memory size for 100 tasks

5.2 Changing the Number of Iterations

These experiments were done on 900 tasks with lengths of 4000 to 8000. It is well known that the number of examined solutions by DAHS algorithms is increased by raising the number of iterations which will lead to increased accuracy of the algorithms and provision of the most solution. First, the experiment was done assuming that the sliding window size was 10 and its obtained results are illustrated in Fig. 4. The following figure shows that the value of makespan varies less than 2% after 100 iterations.

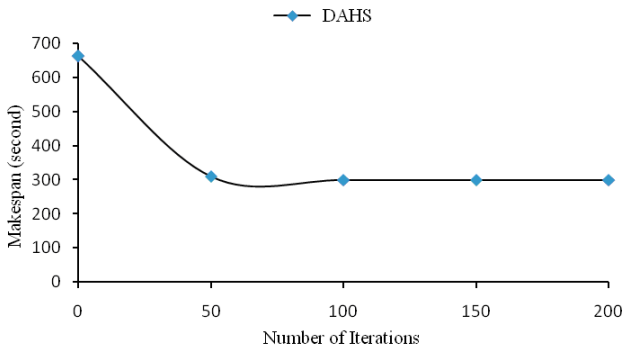

Fig. 4 Makespan of DAHS algorithm versus number of iterations for 900 tasks having length of 4000 to 8000 and sliding window size of 10

In order to select the best size for sliding window, the same experiment was repeated for sizes of 20 and 30. The obtained results are shown in Fig. 5 and Fig. 6. As can be seen in the following figures, the value of makespan is set to 2% of its final value for sliding window size of 20 after 400 iterations and for sliding window size of 30 after 800 iterations.

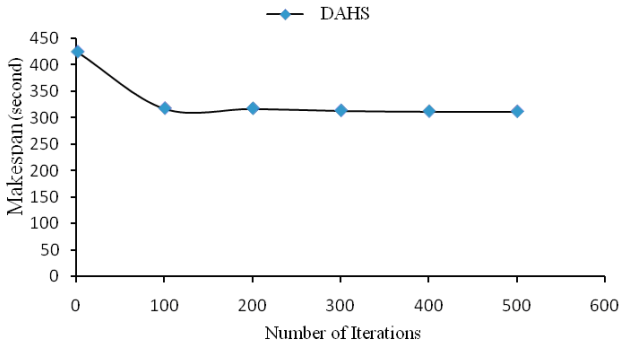

Fig. 5 Makespan of DAHS algorithm versus number of iterations for 900 tasks having length of 4000 to 8000 and sliding window size of 20

Investigation on the results obtained from three experiments show that at small number of iterations (lower than 50), the large sliding window size has produced much better results. However, at large number of iterations, the best answer seems to occur for the smaller sliding window

size. In further experiments, the sliding window size is decreased and the number of iterations is increased to reach more appropriate solutions. Thus, it is recommended to take the sliding window size and the number of iterations equal to 10 and 100, respectively.

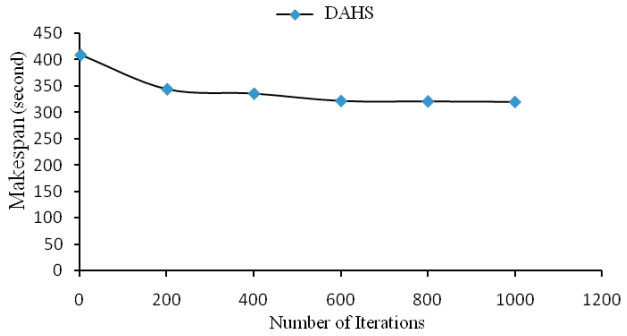

Fig. 6 Makespan of DAHS algorithm versus number of iterations for 900 tasks having length of 4000 to 8000 and sliding window size of 30

5.3 Changing the Number of Tasks

The number of tasks is varied from 100 to 1100 and the length of tasks has been assumed in the range of 4000-8000. Makespan, mean square deviation of load, and average utilization of resources have been shown in Fig. 7, Fig. 8 and Fig. 9 for a specific number of tasks out of 20 runs of the algorithm. It can be observed that performance of the algorithm is increased by raising the number of tasks and this algorithm attains a good extendibility.

Performance of DAHS algorithm has been compared versus that of DGA algorithm. The results uncover that DAHS algorithm has smaller makespan in all cases. Mean square deviation of DAHS is reported 2% smaller than DGA, while average utilization of it is 2.73% higher than that of DGA.

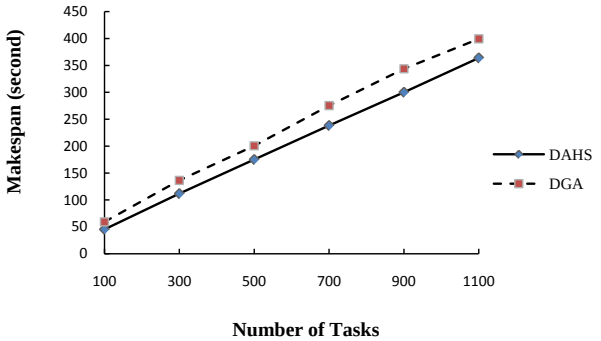

Fig. 7 Makespan versus number of tasks for DAHS and DGA algorithms

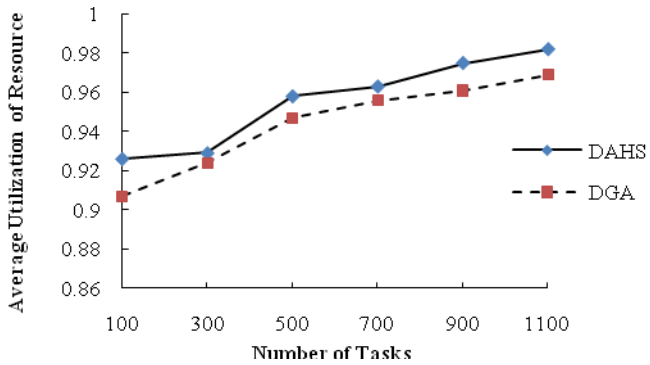

Fig. 8 Average utilization of resources versus number of tasks for DAHS and DGA algorithms

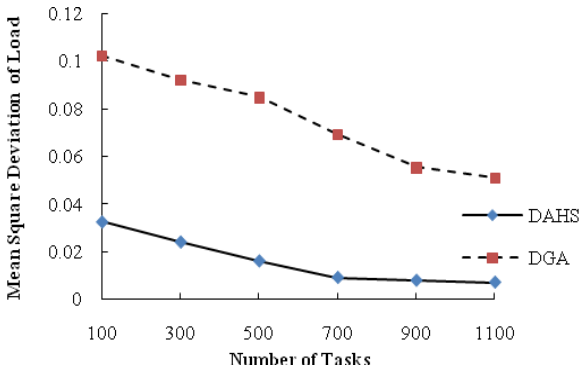

Fig. 9 Mean square deviation of load versus number of tasks for DAHS and DGA algorithms

5.4 Changing the Length of Tasks

Makespan, mean square deviation of load, and average utilization of resources have been evaluated for 300 tasks with a length of 4000 to 8000. Generally speaking, the execution time of tasks is increased at longer lengths of them and choosing inappropriate is expected to incur great variations in the load of resources. A look on Fig. 10 implies that increasing the lengths of tasks will naturally raise scheduling length. Therefore, in order to assess performance of the algorithm, mean square deviation of load and average utilization of resources must be examined (Fig. 11 and Fig. 12). It can be seen that DAHS algorithm reduces the mean square deviation 3.24% in comparison with DGA algorithm, while raises the average utilization of resources for about 1.8%. Results of simulation reveal that DAHS algorithm is more efficient for smaller tasks, so one should increase the number of iterations to attain better time solutions with longer tasks.

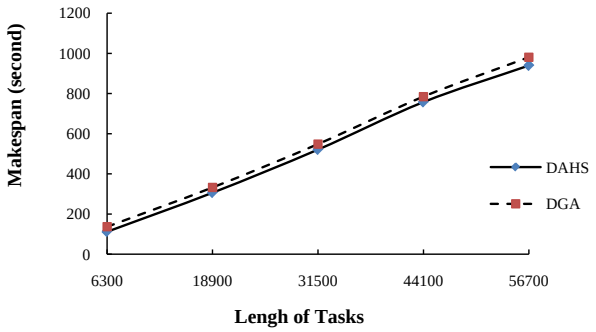

Fig. 10 Makespan versus length of tasks for 300 tasks in DAHS and DGA algorithms

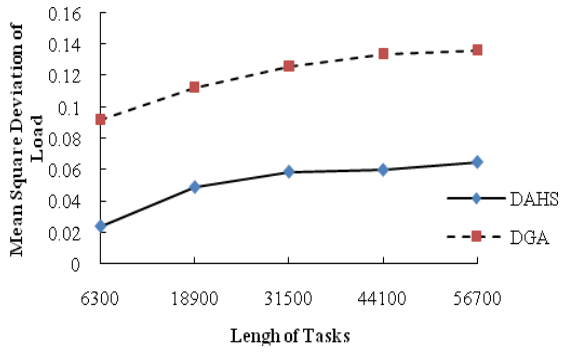

Fig. 11 Mean square deviation of load versus length of tasks for 300 tasks in DAHS and DGA algorithms

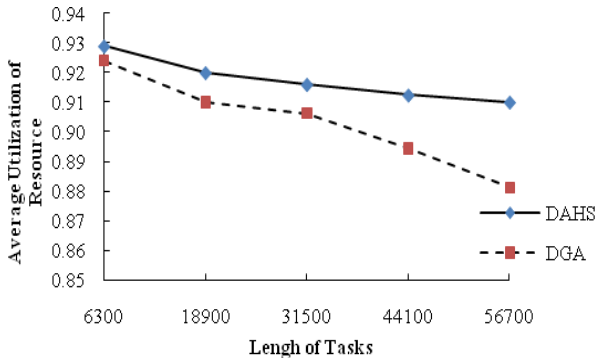

Fig. 12 Average utilization of resources versus length of tasks for 300 tasks in DAHS and DGA algorithms

6. Conclusion

The aim of this paper was to provide an algorithm which was able to create load balancing and grid computing. Harmony algorithms were utilized for this purpose. Parameters of the algorithm were defined as adaptive to improve this algorithm. These parameters are a function of fitness from the previous generation.

When a resource has extra load and a task has been allocated to it in the new solution, this task would be transferred to the underload resource. The results of simulation show that utilization of resources and mean square deviation in DAHS are more than 92% and less than 0.06%, respectively. Comparison of DAHS versus DGA reveals a 10% improvement in makespan as well as a 3% enhancement in utilization.

Future works can concentrate on a combination of the proposed algorithm with the evolutionary algorithms in order to increase rate of convergence. Meanwhile, the current algorithm can be applied to cloud computing by addition of some more specifications.

References

[1] Foster, I., Zhao, Y., Raicu, I., Lu, S., "*Cloud Computing and Grid Computing 360-Degree Compared*," Grid Computing Environments Workshop, pp. 1-10, 2008.

[2] Zeng, J.T., Xia, J.W., Li, J.Z., Li, M.H., "*Multi-objective Optimal Grid Workflow Scheduling with QoS Constraints*," Journal of Advances in Soft Computing, pp. 839-847, 2009.

[3] Hu, Y., Xing, L., Zhang, W., Xiao, M.H., Tang, D., "*A Knowledge-Based Ant Colony Optimization for a Grid Workflow Scheduling Problem*," Lecture Notes in Computer Science, pp. 241-248, 2010.

[4] K. Li, "Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments," *Journal of Systems Architecture*, pp. 1-13, 2007.

[5] Y. Li, Y. Yang, M. Ma, L. Zhou, "A hybrid load balancing strategy of sequential tasks for grid computing environments," *Journal of Future Generation Computer Systems*, Vol. 25, pp. 819-828, 2009.

[6] M. El-Darieby, D.Krishnamurthy, "A Scalable Wide-Area Grid Resource Management Framework", *Proceedings of the International conference on Networking and Services*, pp. 76–84, 2006.

[7] N.G. Shivaratri, P.Krueger, and M.Singhal, "Load Distributing for Locally Distributed Systems," *Computer*, Vol. 25, Issue 12, pp.33-44, 1992.

[8] M. Beltrán and A. Guzmán, "How to balance the load on Heterogeneous cluster," *International Journal of High Performance Computing Applications*, Vol. 23, No. 1, pp. 99–118, 2009.

[9] Zomaya, A.Y., Teh, Y.H.," *Observations on using genetic algorithms for dynamic Load Balancing*," IEEE Transactions on Parallel and Distributed Systems, Vol. 12, Issue 9, pp. 899-912, 2001.

[10] Li, Y., Yang, Y., Zhu, R., "*A Hybrid Load Balancing Strategy of Sequential tasks for Grid Computing Environment*," Future Generation Computer Systems, Vol. 25, Issue 8, pp.819-828, 2009.

[11] Ludwig, S.A., Moallem, A., "*Swarm Intelligence Approaches for Grid Load Balancing*, " Journal of Grid Computing, Vol. 9, No. 3, pp.279-301, 2011.

[12] Heydari, F., Shahhoseini, H.S., "Adaptive *Algorithm for Task scheduling in the Distributed Heterogeneous Systems*

using Harmony Search," The 7th International Conference on Networked Computing (INC), pp.11-16, 2011.

[13] Subrata, R., Zomaya, A.Y. and Landfeldt, B., "*Artificial life techniques for load balancing in computational grids*," Journal of Computer and System Sciences, 73(8):1176-1190, 2007

[14] Lazowska, E. D., Eager, D. L. and Zahorjan, J., "*The limited performance benefits of migrating active processes for load sharing*", *Performance Evaluation Review*, pp. 63-72, 1998.

[15] Geem, Z.W., Kim, J.H. and Loganathan, G.V., *"A new heuristic optimization algorithm: harmony search*," Simulation, Vol. 76, pp.60-68, 2001.

[16] Chen, J., Pan, Q. and Li, H., "*Harmony search algorithm with dynamic subpopulations for scheduling identical parallel machines*," Sixth International Conference on Natural Computation, Vol. 5, pp.2369-2373, 2010.

[17] Mahdavi, M., Fesanghary, M. and Damangir, E., "*An improved harmony search algorithm for solving optimization problems*," Mathematics and Computation, Vol. 188, pp.1567-1579, 2007.

[18] Buyya, R. and Murshed, M., "*Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing*," Journal of Concurrency and Computation: Practice and Experience, Vol. 14, pp.13-15, 2002.

[19] Sulistio, A., Buyya, R., "*A grid simulation infrastructure supporting advance reservation*," Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems, pp. 1-7, 2004.

[20] David Abramson, Rajkumar Buyya, Manzur Murshed, and Srikumar Venugopal. Scheduling parameter sweep applications on global Grids: A deadline and budget constrained cost-time optimisation algorithm. International Journal of Software: Practice and Experience (SPE), 35(5): 491-512, 2005.

[21] Sahoo, B., Mohapatra, S. and Jena, S.K., *"A Genetic Algorithm Based Dynamic Load Balancing Scheme for Heterogeneous Distributed Systems*," Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Vol. 2, pp. 499-505, 2008.

Fatemeh Heydari received her MSEE degrees from the Iran University of Science and Technology (IUST) in 2011. Her research interests include performance evaluation of distributed systems, grid computing and scheduling.

Hadi Shahriar Shahhoseini received his BS degree in electrical engineering from the University of Tehran in 1990, his MS degree in electrical engineering from Azad University of Tehran in 1994, and his PhD degree in electrical engineering from the IUST in 1999. He is an assistant professor of the electrical engineering department in the IUST. His areas of research include networking, supercomputing, and reconfigurable computing. More Than 130 papers have been published from his research works in scientific journals and conference proceedings.