LOW AND HIGH LEVEL HYBRIDIZATION OF ANT COLONY SYSTEM AND GENETIC ALGORITHM FOR JOB SCHEDULING IN GRID COMPUTING

Mustafa Muwafak Alobaedy[1], and Ku Ruhana Ku-Mahamud[2]

[2]*Universiti Utara Malaysia), Malaysia, new.technology@hotmail.com*
[2]*Universiti Utara Malaysia, Malaysia, ruhana@uum.edu.my*

ABSTRACT. Hybrid metaheuristic algorithms have the ability to produce better solution than stand-alone approach and no algorithm could be concluded as the best algorithm for scheduling algorithm or in general, for combinatorial problems. This study presents the low and high level hybridization of ant colony system and genetic algorithm in solving the job scheduling in grid computing. Two hybrid algorithms namely ACS(GA) as a low level and ACS+GA as a high level are proposed. The proposed algorithms were evaluated using static benchmarks problems known as expected time to compute model. Experimental results show that ant colony system algorithm performance is enhanced when hybridized with genetic algorithm specifically with high level hybridization.

Keywords: grid computing, job scheduling, hybrid metaheuristic algorithm, ant colony system, genetic algorithm

INTRODUCTION

Grid computing provides a computing capability to solve complex problems which are not possible to solve using individual resource. Resource Management System (RMS) in grid computing is to schedule jobs and monitor available resources (Kolodziej, 2012). Scheduling in grid computing system is considered as an NP-complete problem. The scheduling algorithm could be based on simple approach such as first come first serve. However, with the increased number of jobs and resources, RMS needs more sophisticated algorithm such as metaheuristic algorithm. There are several metaheuristic algorithms that have been applied in job scheduling problem such as Tabu Search (TS), Genetic Algorithm (GA), Ant Colony System (ACS) and Artificial Bee Colony (ABC). Each algorithm shows good performance in specific instance problem and no algorithm could be concluded as the best algorithm for scheduling algorithm or in general, for combinatorial problems (Yang, 2014). Hybrid approach between one or more algorithms will combine the advantages of individual algorithms (Kolodziej, 2012).

ACS algorithm is one of the prominent metaheuristic algorithm solving various types of combinatorial problems (Dorigo & Stutzle, 2004). The three main phases in ACS are the ants' solution construction, global pheromone trail update, and local pheromone trail update. ACS algorithm starts solution construction when the ant moves from node to node and will choose the node using one of the two rules. The first rule is called *pseudorandom proportional* rule which is based on exploitation mechanism. The second rule uses exploration mechanism which is based on the probability distribution used in Ant System (AS). The tuning between

exploitation and exploration is controlled via a parameter fixed by the user. ACS algorithm applies the local and global pheromone trail update. All ants apply a local pheromone update rule immediately after moving on arcs during the tour construction using the evaporation concept. In global update, only one ant (the best-so-far ant) is allowed to add pheromone after all ants have finished constructing their tours.

GA is a well-known algorithm to solve various types of combinatorial optimization problems developed in 1975 by John Holland (Blum & Roli, 2003). It is applied in various types of scheduling problems, such as manufacturing scheduling, scheduling of production and transport systems, and scheduling workflow applications in cloud environment. GA has three prime operators, namely crossover, mutation, and selection. The solution quality produced by GA depends upon many factors, such as the initializing of the population method, the type of crossover, mutation, and replace methods.

This study has implemented low and high level of hybridization between ACS and GA. The proposed hybrid algorithms are called ACS(GA) for low level and ACS+GA for high level. The rest of the paper is organized as follows. Next section presents the problem formulation and the benchmark for static grid scheduling. The implementation of hybrid ACS and GA in grid computing is described in the subsequent section followed by a section on results of the experiments conducted on ACS hybrid with GA in grid computing. Finally, the conclusion and future work are provided.

PROBLEM FORMULATION

This study has considered a static grid computing system based on batch mode. In batch mode, the jobs are grouped into batches and each batch is assigned to the resources via the scheduler. Jobs are independent and job size is expressed in Million of Instruction (MI) while the resource capacity is expressed in Million of Instruction Per Second (MIPS).

One of the successful models for heterogeneous static computing system is ETC (Braun et al., 2001). ETC model arranges the information in a two dimension matrix called ETC matrix. Each entry in the matrix $ETC[i,j]$ represents the expected execution time of job$[i]$ on machine$[j]$. In ETC matrix, the elements along a row represent the estimates of the expected execution times of a given job on different machines, while the elements along a column give the estimates of the expected times of different jobs on a given machine. The time required to process a task on a resource is calculated as follows:

$$ETC[i,j] = [task_i] \,/\, [machine_j] \tag{1}$$

$ETC_{n\times m}$ is a matrix with two dimensions $n \times m$ where n is the number of jobs and m is the number of machines. The $[task_i]$ and $[machine_j]$ values are generated using range-based technique (Braun et al., 2001). The ETC matrix is categorized into four categories i.e. high job heterogeneity and high machine heterogeneity, high job heterogeneity and low machine heterogeneity, low job heterogeneity and high machine heterogeneity and low job heterogeneity and Low machine heterogeneity. Each category will be classified further into three classes: consistent, inconsistent, and semi-consistent ETC matrices. These classes are orthogonal to the previous categories. This combination produced twelve ETC matrices. In ETC matrix, each machine has a load to process before processing the new jobs. The previous load expressed using ready time vector. The ready time vector of all machines is defined as:

$$ready\_time = [ready_1, ready_2, \dots, ready_m] \tag{2}$$

The completion time of $machine_j$ is calculated using:

$$completion[j] = ready_j + \sum_{i \in Task(j)} ETC[i,j], \tag{3}$$

where $Task(j)$ is the set of jobs assigned to the $machine\ j$. The $completion[j]$ parameters are the coordinates of the following completion vector:

$$completion = \left[completion[1], \ldots, completion[m]\right]^T \tag{4}$$

Using completion vector, the makespan is calculated using the following equation:

$$makespan = \max_{j \in M} (\text{completion[j]}) \tag{5}$$

where M is the number of machines (Kolodziej, 2012). Makespan metric is considered as the main metric to measure the grid computing performance which in turn measures the scheduling algorithm efficiency.

PROPOSED HYBRID ACS AND GA ALGORITHMS

Hybridization could be between any types of algorithms such as heuristic and metaheuristics algorithms (Talbi, 2013). There are two levels of hybridization, namely low and high levels. In low level hybridization, the algorithms interchange their inner procedures. The low level hybridization could be presented as $Algorithm_1(Algorithm_2)$, where $Algorithm_1$ is the main algorithm and $Algorithm_2$ is the subordinate algorithm. On the other hand, high level hybridization could be represented as $Algorithm_1 + Algorithm_2 + \cdots + Algorithm_n$ where $Algorithm_1$ will start first, and then it will call $Algorithm_2$ after it finishes its process and so on.

The proposed low level hybridization called ACS(GA) has ACS as the main algorithm which during its flow will call the GA for enhancement. ACS(GA) will refine the solution produced by each ant in ACS. The best solution produced by the ants is sent to GA for enhancement. The enhanced solution is returned to the ant to update the pheromone value. Therefore, the ant will update the pheromone using the enhanced solution which will increase the pheromone value. The pheromone value will influence the movements of the ants in the next iteration. Figure 1 represents the pseudocode for ACS(GA) algorithm. ACS(GA) algorithm is different from the one presented by Liu, Chen, Dun, Liu, & Dong (2008) which is based on using GA to choose, cross, and mutate the parameters of ant colony algorithm. In the proposed ACS(GA) algorithm, GA is used to enhance the best-so-far solution found by ants in every cycle as shown in Figure 1 with the step **"Add (best ant solution from ACS to P)**".

A high level hybridization algorithm called ACS+GA is proposed to enhance the solution produced by ACS algorithm. ACS will start to generate a good quality solution as one of the initial population for GA which will then be enhanced by GA. The algorithm notation ACS+GA means ACS starts first followed by GA. ACS algorithm execution and pheromone update are totally independent of GA and vice versa. Figure 2 depicts the high level pseudocode hybridization of ACS+GA algorithm. High level hybridization between ACO and GA has been proposed by Kolasa & Krol (2010) for the assignment problem. In each cycle of the algorithm, the best solution of the two algorithms is selected and the search is continued by both of them. In contrast, the proposed ACS+GA algorithm applies the GA to refine the solution found by ACS. In other words, the ACS starts with a specific number of iterations or a period of time. Then the solution found by ACS is passed to GA as one of the initial population. GA will refine the solution received from ACS.

Procedure ACS(GA)
Step 1- Initialize the number of ants *n*;
Step 2- Initialize parameters and pheromone trails;
Step 3- While (Termination condition not met) Do;
Step 4- For i = 1 to *n* Do;
Step 5- Construct new solution;
Step 6- Apply local pheromone update;
Step 7- End For;
// Genetic algorithm starts here;
Step 8- Initialize population (*P*);
Step 9- **Add (best ant solution from ACS to *P*);**
Step 10- Evaluate (*P*);
Step 11- While (termination condition not met);
Step 12- Ṕ← Select (*P);*
Step 13 - Crossover (Ṕ);
Step 14- Mutate (Ṕ);
Step 15- Evaluate (Ṕ);
Step 16- *P* ← Replace (Ṕ ∪ P);
Step 17- End While;
// Genetic algorithm ends here;
Step 18- Apply Global pheromone update;
Step 19- Update best found solution s^*;
Step 20- End while;
Step 21- Return the best solution;
End Procedure;

Figure 1. ACS(GA) Pseudocode

Procedure ACS+GA
Step 1- Initialize the number of ants *n*;
Step 2- Initialize parameters and pheromone trails;
Step 3- While (Termination condition not met) Do;
Step 4- For i = 1 to *n* Do;
Step 5- Construct new solution;
Step 6- Apply local pheromone update;
Step 7- End For;
Step 8- Apply Global pheromone update;
Step 9- Update best found solution s^*;
Step 10- End while;
// Genetic algorithm starts here;
Step 11- Initialize population (*P*);
Step 12- **Add (best ant solution from ACS to *P*);**
Step 13- Evaluate (*P*);
Step 14- While (termination condition not met);
Step 15- Ṕ← Select (*P);*
Step 16 - Crossover (Ṕ);
Step 17- Mutate (Ṕ);
Step 18- Evaluate (Ṕ);
Step 19- *P* ← Replace (Ṕ ∪ P);
Step 20- End While;
// Genetic algorithm ends here;
Step 21- Return the best solution;
End Procedure;

Figure 2. ACS+GA Pseudocode

EXPERIMENTS AND RESULTS

The proposed hybrid algorithms ACS(GA) and ACS+GA have several parameters need to be tuned in order to achieve the desired performance. Therefore, this study adopted the best values from the literature. The parameters values for ACS and GA were selected based on recommended values from Dorigo & Stutzle (2004) and Xhafa, Barolli, & Durresi (2007) respectively. Table 1 presents the parameters value for ACS and GA.

Table 1. ACS and GA Parameters

ACS	Beta	Evaporation rate	No of ants	q	Run time
	8	0.6	10	0.9	45 second (high level) 2 seconds (low level)
GA	Population	Intermediate	Crossover	Mutation	
	10	6	0.9	0.4	

In addition to parameters value, GA has several types of operators. Table 2 shows the types of operators implemented in this study which are adopted from Xhafa et al. (2007).

Table 2. GA Implemented Operators

Elitism	Selection operator	Crossover operator	Mutation operator
True	Tournament = 3	Fitness based	Re-balanced

Experiments have been conducted using Intel® Core (TM) i7-3612QM CPU @ 2.10GHz and 8G RAM. The grid computing simulator is developed using visual C#. Each algorithm was executed 10 times in order to calculate the average values as well as to get the best run.

Tables 3 and 4 present the experimental results. The first column of each table represents the instance name with an abbreviation code: *x-yyzz* as: *x* represents the type of consistency; *c* means consistent, *i* means inconsistent, and *s* means semi-consistent. *yy* represents the heterogeneity of the jobs; *hi* means high and *lo* means low. *zz* represents the heterogeneity of the machines; *hi* means high and *lo* means low. For example: *c_hilo* means consistent environment, high heterogeneity in jobs and low heterogeneity in machines.

The proposed hybrid algorithms were compared with GA and ACS algorithms in terms of best makespan value. Table 3 shows that the proposed ACS+GA algorithm outperforms other algorithms for eight instances out of twelve. The proposed ACS(GA) algorithm achieved good results only on two instances followed by GA on two instances as well. In terms of average makespan value, the proposed algorithm ACS+GA produced the best performance on six instances out of twelve followed by GA on five instances and ACS on one instance as show in Table 4. In order to represents the overall results of the proposed algorithms visually, the geometric mean is used to normalize the makespan values of the 12 instances.

Table 3. Best Makespan Values

Instance	GA	ACS	ACS(GA)	ACS+GA
c_hihi	11215488.9	10794610.8	10672485.1	**10533616.4**
c_hilo	182232.0	179762.4	**179200.7**	180289.8
c_lohi	374686.0	346838.4	352392.8	**345233.3**
c_lolo	6138.5	6051.8	6045.6	**6001.9**
i_hihi	3995843.4	4066163.7	4125479.4	**3924281.6**
i_hilo	91682.3	93829.0	**91674.8**	91709.9
i_lohi	**134151.1**	137176.5	138843.6	134796.3
i_lolo	**3045.3**	3209.0	3194.6	3164.3
s_hihi	6223749.5	6119602.0	6147994.6	**5854357.3**
s_hilo	120447.3	120539.1	120885.9	**119123.9**
s_lohi	181155.5	178584.8	172357.1	**172225.0**
s_lolo	4246.4	4350.4	4293.5	**4225.7**

Table 4. Average makespan values

Instance	GA	ACS	ACS(GA)	ACS+GA
c_hihi	11266455.7	10947366.9	10892151.9	**10849427.3**
c_hilo	183264.9	181434.4	181019.7	**180970.8**
c_lohi	375322.2	**353670.8**	355078.0	353882.8
c_lolo	6152.5	6120.0	6114.3	**6074.3**
i_hihi	**4029108.7**	4261681.8	4261742.3	4115442.3
i_hilo	**91682.3**	94832.7	94420.9	93514.0
i_lohi	**135625.0**	144178.5	143256.2	138746.9
i_lolo	**3051.0**	3280.0	3243.5	3232.7
s_hihi	6317823.2	6322969.8	6269617.2	**6119177.6**
s_hilo	120664.4	122440.4	122341.2	**120576.8**
s_lohi	181734.6	181737.4	180022.2	**177965.1**
s_lolo	**4249.9**	4399.4	4349.0	4326.3

Figures 3 and 4 present the geometric mean of best and average makespan values respectively. Figure 3 shows that the proposed ACS+GA outperforms other algorithms followed by ACS(GA), ACS, and GA for geometric mean of the best makespan value. In terms of geometric mean of the average makespan value, the proposed algorithm ACS+GA achieved the best performance followed by GA, ACS(GA), and ACS as shown in Figure 4.

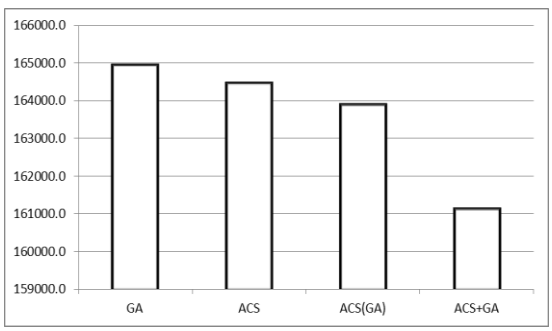

Figure 3. Geometric Best Makespan

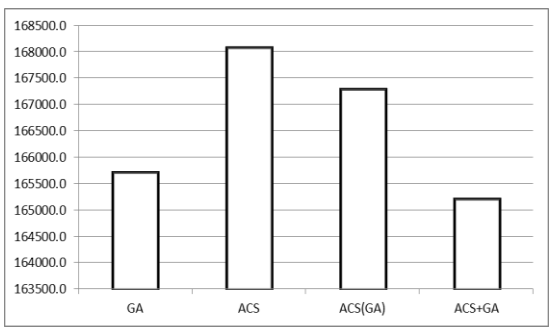

Figure 4. Geometric Average Makespan

CONCLUSION AND FUTURE WORK

Comparison between low and high level hybridizations of ACS and GA algorithms have been performed for job scheduling problems in static grid computing environment. The high level hybridization mode of ACS and GA algorithm produced better performances in terms of best and average makespan values for job scheduling. Future work related to ACS+GA algorithm could be implemented to solve the job shop scheduling. In addition, other local search algorithms such as TS and SA could by hybridized with ACS to solve the job scheduling problem in grid computing.

ACKNOWLEDGMENT

The authors wish to thank the Ministry of Higher Education Malaysia for funding this study under the Fundamental Research Grant Scheme, S/O codes 12819 and 11980, and RIMC, Universiti Utara Malaysia, Kedah, for the administration of this study.

REFERENCES

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *Journal of ACM Computing Surveys*, 35(3), 268–308. doi:10.1145/937503.937505

Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., … Freund, R. F. (2001). A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Jobs onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6), 810–837. doi:10.1006/jpdc.2000.1714

Dorigo, M., & Stutzle, T. (2004). *Ant Colony Optimization*. Cambridge, Mass: MIT Press.

Kolasa, T., & Krol, D. (2010). ACO-GA Approach to Paper-Reviewer Assignment Problem in CMS. *Proceedings of the International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, 360–369. Gdynia, Poland.

Kolodziej, J. (2012). *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*. New York: Springer. doi:10.1007/978-3-642-28971-2

Liu, J., Chen, L., Dun, Y., Liu, L., & Dong, G. (2008). The Research of Ant Colony and Genetic Algorithm in Grid Job Scheduling. *Proceedings of the International Conference on MultiMedia and Information Technology*, 47–49. Three Gorges: IEEE.

Talbi, E. (2013). A Unified Taxonomy of Hybrid Metaheuristics with Mathematical Programming, Constraint Programming and Machine Learning. In E. Talbi (Ed.), *Hybrid Metaheuristics*. Heidelberg: Springer.

Xhafa, F., Barolli, L., & Durresi, A. (2007). An Experimental Study on Genetic Algorithms for Resource Allocation on Grid Systems. *Journal of Interconnection Networks*, 8(4), 427–443.

Yang, X.-S. (2014). *Nature-Inspired Optimization Algorithms*. Amsterdam: Elsevier.