ADAPTIVE INTELLIGENCE JOB ONLINE SCHEDULING WITHIN DYNAMIC GRID ENVIRONMENT BASED ON GRIDSIM

Siriluck Lorpunmanee[1], Mohd Noor Md Sap[2], Abdul Hanan Abdullah[3]

[1]Faculty of Science and Technology,
Suan Dusit Rajabhat University,
295 Rajasima Road, Dusit, Bangkok, Thailand

[2,3]Faculty of Computer Science and Information Systems,
University Technology of Malaysia,
81310 Skudai, Johor, Malaysia

Email: [1]siriluck_lor@dusit.ac.th, [2]mohdnoor@fsksm.utm.my, [3]hanan@fsksm.utm.my

Abstract: This paper concentrates on the design and implement of the grid system for study of adaptive job scheduling algorithm based on GridSim. The common problems of job scheduling in grid system like heterogeneous of jobs, resources and dynamic an arrival time of new jobs significantly changes, can be deal with this solution. The idea behind the adaptive job scheduling algorithm is the hybrid algorithms that consist of Ant Colony Optimization (ACO) and Tabu algorithms. Additionally, the provided common information from Grid Information Service (GIS) and an arrival new job are calculated by Fuzzy C-Means (FCM) algorithm in order to evaluate the current status of resources and groups of arrival jobs. Moreover, both dynamic and static information are handled by the solution. In static case, the resource information such as a number of CPUs of a machine, CPU speed, a number machine in the grid system is significantly known in advance while dynamic information like the arrival jobs that are submitted to the system any time during simulation. In the results, this paper shows the comparison results between the adaptive job scheduling algorithms and the traditional algorithms.

Keywords: Ant Colony Optimization algorithm, Tabu algorithm, Fuzzy C-Means algorithm, Grid Information Service, online job, Adaptive scheduling algorithm.

1. INTRODUCTION

Computation Grid technologies [1] are a new trend and appear as a next generation of the distributed heterogeneous system. It combines physical dynamic resources and various

applications. Currently, it is a great potential technology that leads to the effective utilization of the resources. In particular, the complex problems such as scientific, engineering and business need to utilize the huge resources and the performance potential of computation grid. Therefore, the performance of grid system is the key to success. Scheduling is the main point behind these successes.

Effective job scheduling in grid requires to model grid resources and computation requests of jobs, determine the current workload of each grid nodes, and evaluate the execution time of job before allocating to appropriate resource as well. The goals are to achieve the best performance and load balancing across the different domain system. However, when applying those methods to grid environment, the results often happen to be poor performance within heterogeneous grid resources. Whereas, grid environment is open and dynamic, the jobs often arrive at grid in various load and types. For instance, the scheduling needs time slot to schedule a job within various workloads. Hence, the adaptive grid scheduling is the desired algorithm to handle different jobs workload and also various load of grid resources. One of the most important methods of a Grid Resources Management System (GRMS) is currently capable to allocate jobs to grid resources. The realistic grid system, it is usually be impossible for users and grid system administrators to manually find and specify all the needed grid resources during the arrival of jobs in fact. In the dynamic nature of grid, the various load of grid resources availability can constantly change at the moment. Therefore, the requirement of an automatic method to allocate jobs to grid resources is a critical. This method is generally provided by scheduler. Typically, it is difficult to achieve this point because the scheduling problem is defined as NP-hard problem [2] and it is not trivial.

Traditional and heuristic scheduling algorithms [1, 3-5] are often proposed in heterogeneous computing environment. These algorithms focus on explicit constraints and static information of system load to schedule jobs. The dynamic grid environment, the system workload of each grid nodes cannot be determined in advance. Therefore, grid scheduling desires an adaptive scheduling algorithm.

The motivation of this paper is to develop a grid scheduling algorithm that can perform efficiently and effectively in terms of grid efficiency, make-span time and job tardiness. Not only does it improve the overall performance of the system but it also adapts to the dynamic grid system. First of all, this paper proposes an Ant Colony Optimization (ACO) algorithm to find the optimal resource allocation of each job within the dynamic grid environment. Secondly, Tabu search algorithm is used to adjust performance of grid system because online jobs are often submitted to grid system. Third, the simulation of the experiment is presented. This simulation is an extension of GridSim [6] toolkit version 4.0, which is a popular discrete-event simulator and grid scheduling algorithm. The simulator

defines the different workload of resources, the arrival time of independent jobs, the size of each job, the criteria of a scheduler, etc. Finally, this paper discusses about this idea for dynamic grid environment within fully controlled conditions [7].

The rest of the paper is organized as follows. In section 2, the literature review is presented. Section 3 describes the Grid simulation, job scheduling design, and briefly illustrates how to implement an Adaptive intelligent scheduling system into GridSim toolkit simulator. Section 4 illustrates the experiment set and the comparison between this solution and the other algorithms. Section 5 concludes the paper. Finally, Section 6 suggests the future works.

2. LITERATURE REVIEW

In the past few years, researchers have proposed scheduling algorithms for parallel system [8-12]. However, the problem of grid scheduling is still more complex than the proposed solutions. Therefore, this issue attracts the interest of the large number of researchers [3, 13-16].

Current system [17] of grid resource management was surveyed and analyzed based on classification of scheduler organization, system status, scheduling and rescheduling policies. However, the characteristics and various techniques of the existing grid scheduling algorithms are still complex particularly with extra components.

At the present time, job scheduling on grid computing aims not only to find an optimal resource to improve the overall system performance but also to utilize the existing resources more efficiently. Recently, many researchers have been studied several works on job scheduling on grid environment. Those studies are the popular heuristic algorithms, which have been developed, are min-min [5], the fast greedy [5], tabu search [5], Genetic algorithm [18] and an Ant System [4].

The heuristic algorithms proposed for job scheduling in [5] and [4] rely on static environment and the expected value of execution times. Whereas, [18] proposed Genetic algorithm based on static jobs and static load of each node, while the realistic grid environment, in which the system load of grid nodes always change during the deployment of jobs in fact.

H. Casanova et al. [19] and R. Baraglia et al. [20] proposed the heuristic algorithms to solve the scheduling problem based on the different static data, for example, the execution time and system load. Unfortunately, all of information such as execution time and workload cannot be determined in advance of dynamic grid environments.

In 1999, the Ant Colony Optimization (ACO) meta-heuristic was proposed by Dorigo, Di Caro and Gambardella, which has been successfully used to solve many NP-

problem, such as TSP, job shop scheduling, etc. In the past few years, several researchers proposed solutions to solve grid scheduling problem [21] by using ACO.

Several studies have been trying to apply ACO for solving grid scheduling problem. Z. Xu et al. [22] proposed a simple ACO within grid simulation architecture environment and used evaluation index in response time and resource average utilization. E. Lu et al. [23] and H. Yan et al. [24] also proposed an improved Ant Colony algorithm, which could improve the performance such as job finishing ratio. However, they have never used the various evaluation indices to evaluate their algorithm.

The ACO becomes very popular algorithm to apply for solving grid scheduling problem. However, most of the mentioned algorithms were not taken into consideration some evaluation indices, for example, grid efficiency, makespan time, average waiting time and total tardiness time that has been shown in [25].

3. GRID SIMULATOR AND JOB SCHEDULING DESIGN

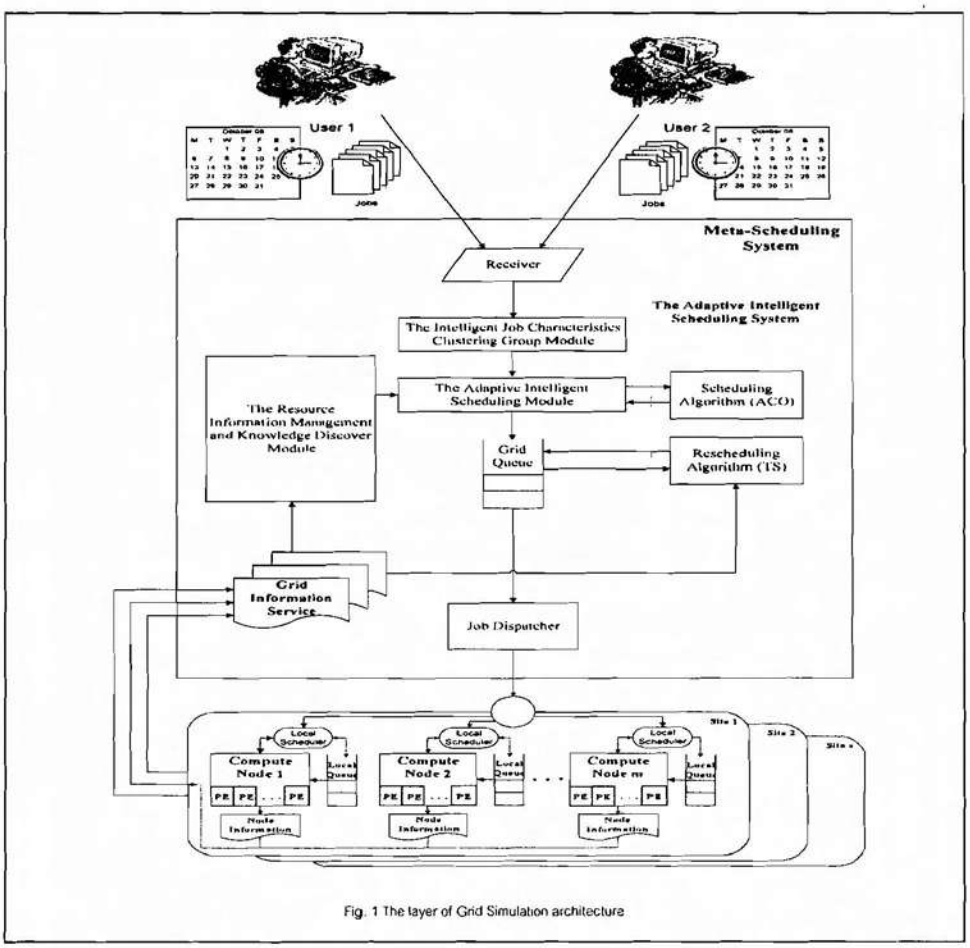

Fig. 1 The layer of Grid Simulation architecture

3.1 GRID SIMULATION ARCHITECTURE

The grid environment is a complex heterogeneous system consisting of many organization domains or site in which no one has full control of all available resources and grid applications (jobs). For this reason, the simulation of these systems is complicated. The simulator presented in this research is based on GridSim [6] and [7]. It also extends internal entities to more complex requirements. Additionally, the architecture is comprised the distributed grid nodes, local schedulers with local queues, and grid Meta-scheduler system with grid queue and job dispatcher. Moreover, the main idea behind an adaptive intelligent scheduling system comprised the clustering of job and grid resource, scheduling algorithm with rescheduling algorithm is proposed within this simulator.

In this model, the layer of all simulation is depicted in Fig 1. The users can register any time in the grid environment, after that they are able to submit jobs to the grid environment. The submitted jobs are received at the Receiver side. The receiver side is to automatically submit the selected jobs to the Adaptive Intelligent Scheduling System. Where, the receiver has the function to maintain the job work load, user name, etc. Similarly, the registration of distributed system is required in grid system in order to be grid resources and their information are gathered at Grid Information Service (GIS). At the Meta-Scheduling System, all of the static and dynamic information about the system state in the grid system such as CPU speed, CPU load, number of CPUs in the grid resources, etc. has been collected at the Grid Information Service (GIS). This process is similar to GRIS (Grid Resource Information Server) registering with GIIS (Grid Index Information Server) in Globus toolkit [6]. Their information is then used to calculate the optimal resources for processing the job. Therefore, the function of the Grid Information Service (GIS) is to maintain and update the status of the grid resources. For each site, the sensor updates the local information by first querying the local information system and then updating their owner information in GIS. For dynamic information about current state of the system, is usually changed and needs for updating each state change in GIS, for example an amount of executing job on CPUs and waiting in the grid queue and also a number of free CPUs are calculated based on their information. Moreover, the expected time complete of each job in the system is estimated. Therefore, this model develops the mechanism as the Evaluation Resource Performance Method to notify and their information are used for making scheduling decision, that is, GIS is extended in order to prepare each significant resource states information. As the local policy, it is applied to each machine that defines their access rights. This policy is applied through a local resource management system for example.

Before allocated the job to availably proper grid resource, the job is calculated to the group based the similar degree of job characteristics by applying the Intelligent Job

Characteristics Clustering Group Module. At the same time, each grid resource as machine is grouped based the similar degree of its characteristics by applying the Resource Information Management and Knowledge Discover Module. In this module, the important current information state of each grid node is provided by GIS and also provided by the Evaluation Resource Performance Method. Once, the job and grid node have been already grouped after that the job is allocated to availably proper grid node by the Adaptive Intelligent Scheduling Module. The mechanism is the job allocated to grid node based its group, for example the light job load should be executed in low performance of grid node, and in contrast, the heavy job load should be executed in high performance of grid node as well. Whenever, the grid queue is not empty, that is, some grid node that has been assigned for the new submitted job is processing the job and have no enough time slot for new submitted job from grid queue. Therefore, rescheduling method is proposed based the current information state of the grid node to solve this issue.

The purpose of a Job Dispatcher is to deliver jobs to a selected grid resource which jobs are in grid queue. The assumption is a close affinity between GIS and grid resources information. Moreover, Meta-Scheduling system gathers the local schedule information via GIS for making schedule decision. In fact, the job stays in the local queue before beginning execution on the local system and the job is computed based on the local scheduling policy. The purpose of a job dispatcher is to deliver jobs to a selected resource. Finally, all ideas mentioned above are the support evidence of this simulator and including the Adaptive Intelligent Scheduling System.

3.2 UML diagram of Grid Simulation Model

In this simulation, each simulated system such as grid resources, Meta-scheduler and users, that interacts with each other is referred to as an entity. An entity operates in parallel in its own thread. In Fig 3, it shows the class relationship among each other, the specification of each class comprises three parts: name, attributes, and methods. In the class diagram, attributes and methods are prefixed with characters '+' and '-' show access modifiers public and private respectively. The simulation implements the following classes.

① class *InitialSetup*: it is main class for the starting of this simulation, performing the whole experiment and gathering the results at the end. It generates initial parameter values and all entities such as grid resources, *Metascheduler* and users by applying from the method *createGridResource* and *createJobSubmissionSystem*. In *createGridResource* method, it creates grid resource and its properties. In this simulation, a machine and its properties such as system architecture, Operating System (OS), time zone, and local policy (time or space shared) are generated to form a resource. Moreover, each grid resource contains one or more

machines within its properties. Similarly, a machine contains one or more PEs (Processing Elements or CPUs) within the same MIPS rating. Upon creating an entity with a specified the centralized scheduler, this class creates a new instance of the *Metascheduler* class in order to maintain all arrived jobs, grid resources status and to perform the Adaptive Intelligent Scheduling system. As *createJobSubmissionSystem* method, its function generates the jobs within its characteristics by creating a new instance of *JobSubmissionSystem* class.

(2) class *GridResource*: it is provided by *GridSim* class. The simulation creates a new instance of *GridResource* class in order to simulate a resource with its properties. The properties of a resource are defined by the entity of *ResourceCharacteristics* class, while a collection of machines is simulated by an instance of *MachineList* class. In addition, *ResourceCalendar* class implements a mechanism to support modeling a local load on grid resource, in which the load of each grid resource may vary according to time zone, time, weekends, and holidays. Besides, each *GridResource* entity contains only one local scheduler of type AllocPolicy class. In this simulation, the *GridResource* only acts as an interface between users and local scheduler, while *AdvancedPolicy* class inherited from *AllocPolicy* class is implemented in order to handle local scheduler and to process submitted jobs. Currently, *GridSim* has *TimeShared* and *SpaceShared* classes for a specific resource-based system. These are also inherited from *AllocPolicy* class and they are implemented by Round Robin and First Come First Serve (FCFS) approaches respectively. In order to simulate the real grid system and the allocation policy for each computational node, therefore, the simple specific resource-based system in *AdvancedPolicy* class exactly like as First Come First Serve (FCFS) is implemented to manage the local queue in the local scheduler. For *AdvancedPolicy* class mechanism, each job is allocated to one Processing Element (PE). If each job requires more than one PE, consequently the local scheduler waits until these PEs are available. The job is returned as FAILED status, the number of available PEs less than the job requirement as shown in Fig 2. In addition, the *body()* method is implemented to handle events communication. That is, it is automatically invoked during operated the grid resource because it is expected to be responsible for internal events communication in *AdvancedPolicy* entity and also sends the events to the other entities.

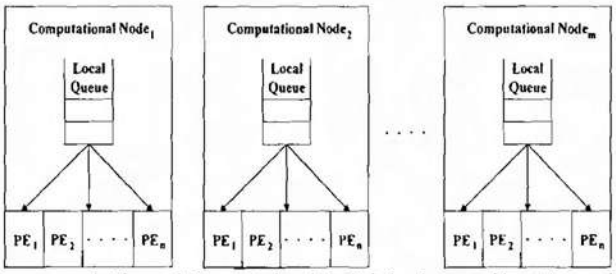

Fig 2: A local queue model for each Computational Node is implemented by *AdvancedPolicy* class in *GridResource* class.

③ class *MetaScheduler*: this class inherited from *GridSim* class is implemented in order to maintain all arrived jobs, to currently calculate all grid resources status such as the number of available CPUs, and their rating and to perform the Adaptive Intelligent Scheduling system. It receives job object (*GridletInfo* object) from *JobSubmissionSystem* entity. The *MetaScheduler* is implemented in centralized scheduler, therefore it gathers information and access to all available grid resources in the system. That is, its function is the selection of the appropriate resource for the job based on all current grid resources status. Using these scheduler algorithms and also the current jobs information, in execution the centralized scheduler is able to approximate various parameters of the current status of scheduler, for example makespan time, expected finish time of the jobs in each grid resource and expected tardiness time before their execution and completion in grid resource. In this research, the Adaptive Intelligent Scheduling system is implemented to determine where to send the job object by applying Fuzzy C-Mean (FCM), Ant Colony Optimization (ACO), and Tabu Search (TS) algorithms. Within these algorithms, they are depicted detail in the next section. The method, namely, *body()* is responsible to handle the events communication among *MetaScheduler*, *JobSubmissionSystem* and *AdvancedPolicy* entities. In the whole events communication, it is asynchronous, that is, each entity is no need to wait for completing the sending or receiving job object. When the appropriate grid resource of the job is selected, the job with the appropriate grid resource is collected in global queue before sending this job object to selected grid resource in *GridResource* entity by applying *dispatchJob* method. Moreover, the grid resource information is updated current status when each completed job object sends back from *AdvancedPolicy* entity by applying *lowerGridletInfoList* method in *ResourceInfo* entity.

④ class *JobSubmissionSystem*: this class inherited from *GridSim* class is implemented in order to generate all jobs by applying *JobGenerator* method. Additionally, it also waits the completed job object that is sent back from *AdvancedPolicy* entity. In *JobGenerator* method, it creates a new instance of the *ComplexGridlet* class equal to number of jobs, in which one *ComplexGridlet* entity represents one job and its properties such as a job owner, an arrival time, a release time, a due date time, a size of job and a job requirement of number Processing Elements (PEs). In *checkStartTimeAndSend* method, it is used for sending the job object (*GridletInfo* object) to the *MetaScheduler* within its current time. Once the completed job object is sent back from *AdvancedPolicy* entity, the *JobReceiver* method is applied to collect all completed jobs and when the end completed job is collected, therefore *sumReportsStatistices* method is generated in order to summarize all information before sending information object to *InitialSetup* entity. Moreover, this *body()* method and the other body() methods are similarly functional in order to handle the events communication.

⑤ class *ComplexGridlet*: this class represents the job and its properties. As its properties, is usually dynamic job information such as an arrival time, a release time, a due date time, a size of job and a job requirement of number Processing Elements (PEs).

⑥ class *GridletInfo*: this class represents the job into the real job during operation the simulation. Since the real job information might change between *JobSubmissionSystem* and *MetaScheduler* entities. It stores various information of real job such as an expected tardiness time of each real job in selected grid resource, an expected finish time of each real job in selected grid resource.

⑦ class *ResourceInfo*: this class is implemented in order to store or prepare dynamic information of each resource for making scheduling decision in *MetaScheduler* entity. That is, it provides several methods to calculate various dynamic information values based on the currently collecting knowledge of each grid resource status, for example expected makespan time and the expected total tardiness time of each grid resource.

⑧ The relationship among a *machinesGroupByFuzzyCMeans*, a *jobsGroupByFuzzyCMeans* and another classes: these are showed the entities relationship each other. Once the simulation is started and the grid resources are registered in the grid system. After that, the users are able to submit the jobs to the grid system. Before each arrival job from grid users is assigned to appropriate grid resource, *machinesGroupByFuzzyCMeans* entity is created new instance by applying *clusterMachineBasedOnPerformance* method in order to arrange the groups of each grid resource. Each collected grid resource information for calculating the groups in *machinesGroupByFuzzyCMeans* entity, the information such as a set of machine in grid resource (*HostID*), the speed of processing (*CPU speed*) of machine (*MIPS*), the comparison between the current expected time complete of the grid system and *m*th machine (*Weight*) and an amount of free CPU in *m*th machine ($FreePEs_m$) are maintained by the *dataSetMachines* entity. Within *machinesGroupByFuzzyCMeans* entity, the Fuzzy C-Means (FCM) algorithm is proposed by using the gathering information in *dataSetMachines* entity.

In *computeFuzzyCMeans* method, is implemented to maintain the FCM algorithm. By all algorithms shown in algorithm [29], can be described in the following classes and methods:

- *computeCenterVectors* method is the algorithm, in which is implemented for calculating the initial value of a cluster center.
- *distanceBetweenTwoObjects* method is the algorithm, in which is implemented for determining the distance between two objects' feature vectors.
- *updateMembershipDegree* method is the algorithm, in which is implemented for updating a objects x's membership degree in the ith cluster.

- *distancePartition* method is the algorithm, in which is implemented to calculate the distance between two partitions.

Finally, the groups of each grid resource are created and its information is gathered by applying *clusterGroupMachines* entity, while a *getClusterMachinesResults* method provides the groups information of machines to *MetaScheduler* entity that is above calculated. Similarly, *jobsGroupByFuzzyCMeans* entity also applies the Fuzzy C-Means algorithm to calculate the groups of the jobs. Additionally, the jobs information such as a number of arrival job in the grid system (*JobID*), the job *j*th owner (*UserID*), the processing requirement of job *j*th in the form of Million Instructions (*MI*) (*LengthJob*), the number of processors (*CPU*) is required by job *j*th (*NumJobPEs*) are applied to calculate within *jobsGroupByFuzzyCMeans* entity. Eventually, *getClusterJobsResults* method provides the groups information of jobs to MetaScheduler entity.

Algorithm 1 [29]: Fuzzy C-Means (FCM)

Step 1: Initialize the membership matrix U and $U^{(0)}$

Step 2: At k-step: calculate the centers vectors $C^{(k)}=[v_j]$ with $U^{(k)}$:

$$v_j = \frac{\sum_{i=1}^{N}(\mu_{ij})^m x_i}{\sum_{i=1}^{N}(\mu_{ij})^m}$$

Step 3: Get the new distance:

$d_{ij} = \|x_i - v_j\|$, $\forall_i = 1,2,...,N$, $\forall_j = 1,2,...,C$;

Step 4: Update the Fuzzy partition matrix (Update $U^{(k)}$, $U^{(k+1)}$)

If $d_{ij} \neq 0$ (means $x_i \neq v_j$)

$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{d_{ij}}{d_{ik}}\right)^{\frac{2}{m-1}}}$$

Else $\mu_{ij} = 1$

Step 5: If $\max||U^{(k+1)} - U^{(k)}|| < \varepsilon$ then STOP; otherwise return to step 2.

⑨ The relationship among a *scheduleACO*, *AntGraph*, *AntColony4Grid*, *Ant4Grid*, *AntColony* and Ant classes: Once the starting to make scheduling decision by applying ACO algorithm in *MetaScheduler* entity, a *scheduleACO* entity is created a new instance by using *addToScheduleACO* method. From referring to the paper in [14, 15], Ant Colony System (ACS) algorithm has been introduced, the foraging behavior of artificial ants colony to cooperate in finding good solutions for difficult optimization problems on graphs similar to the Travel Salesman Problem. In [16], is main framework of Ant Colony System that is extended to play with this simulation. By all algorithms and entities can be described in the following classes and methods:

- *scheduleACO* class is implemented to maintain the ACO algorithm, eventually, each (*the proper machine, job*) pair is added to Global queue. In *initAntGraph* method, constructs the graph of the expected time complete of job on machine, as shown in Fig 4. Additionally, it also creates a new instance of *AntGraph* class in order to maintain the features of a graph.

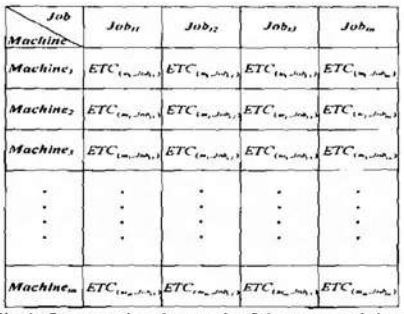

Fig 4: Construction the graph of the expected time complete of job on machine

Note that the expected execution time e_{ij} of jth job on ith machine m_i is defined as the amount of execution time of ith job taken by ith machine m_i, in which ith machine m_i has no load during ith job is assigned. The expected completion time (ETC_{ij}) represents the expected time complete of jth job on ith machine m_i, is defined as the wall-clock time at which ith machine m_i completes jth job after having finished any previously assigned jobs. Therefore, $ETC_{ij} = b_j + e_{ij}$; b_j represents the beginning time of jth job.

- *AntGraph* class is implemented in order to maintain realize the features of a graph for Ant Colony System, that is, provides the methods to update the values, for example heuristic information ($\eta_j(i,m)$), the pheromone trails ($\tau_j(i,m)$). Besides, *resetTau* method calculates the pheromone initialization by applying *predictMinMakeSpanTime* method. In realize ants, they work really in parallel access asynchronously and unpredictably to the graph. In order to handle the concurrent ant access to write or read at the same time, consequently, *AntGraph* entity are implemented in form a synchronization by using the synchronized mechanism of java language.
- *Ant* class is implemented in order to generate artificial ant behaviour. Under the mechanism, each ant works in its own thread in parallel with all others giving to ACO the parallelism shown by actual ant colonies. In this class, the abstract methods like *stateTransitionRule*, *localUpdatingRule*, *compare*, and *end* are implemented and extended into *Ant4Grid* class. The other methods and data members in this class, for example *init()*, *start()*, *run()*,*m_nCurNode*, *m_nStartNode*, and etc. are used internally to perform the activity of the ant.
- *AntColony* class is implemented in order to generate an Ant Colony behaviour. In abstract methods, *createAnts* and *globalUpdatingRule* are implemented and extended into *AntColony4Grid* class. As *createAnts* method, creates the new instances of the *Ant* class that walk on the edges of the graph. The other methods and data members in this class, for example *start()*, *iteration()*, *m_ants*, and etc. are used internally to perform the activity of the whole ant colony.

- *AntColony4Grid* class inherited from *AntColony* class is implemented in order to generate amount of ant into its colony that each ant walks on the edges of the graph (*createAnts* method). Additionally, it also applies *GlobalUpdatingRule* method for updating the pheromones deposited on the arcs when an ant ends it trip.

Ant4Grid class inherited from *Ant* class is implemented in order to maintain a State Transition Rule, that is, brings the artificial ant from a node to another across an arc. As Local Updating Rule, also is performed by using *localUpdatingRule* method, it is used to update the pheromone deposited on the arcs when an ant end its trip.

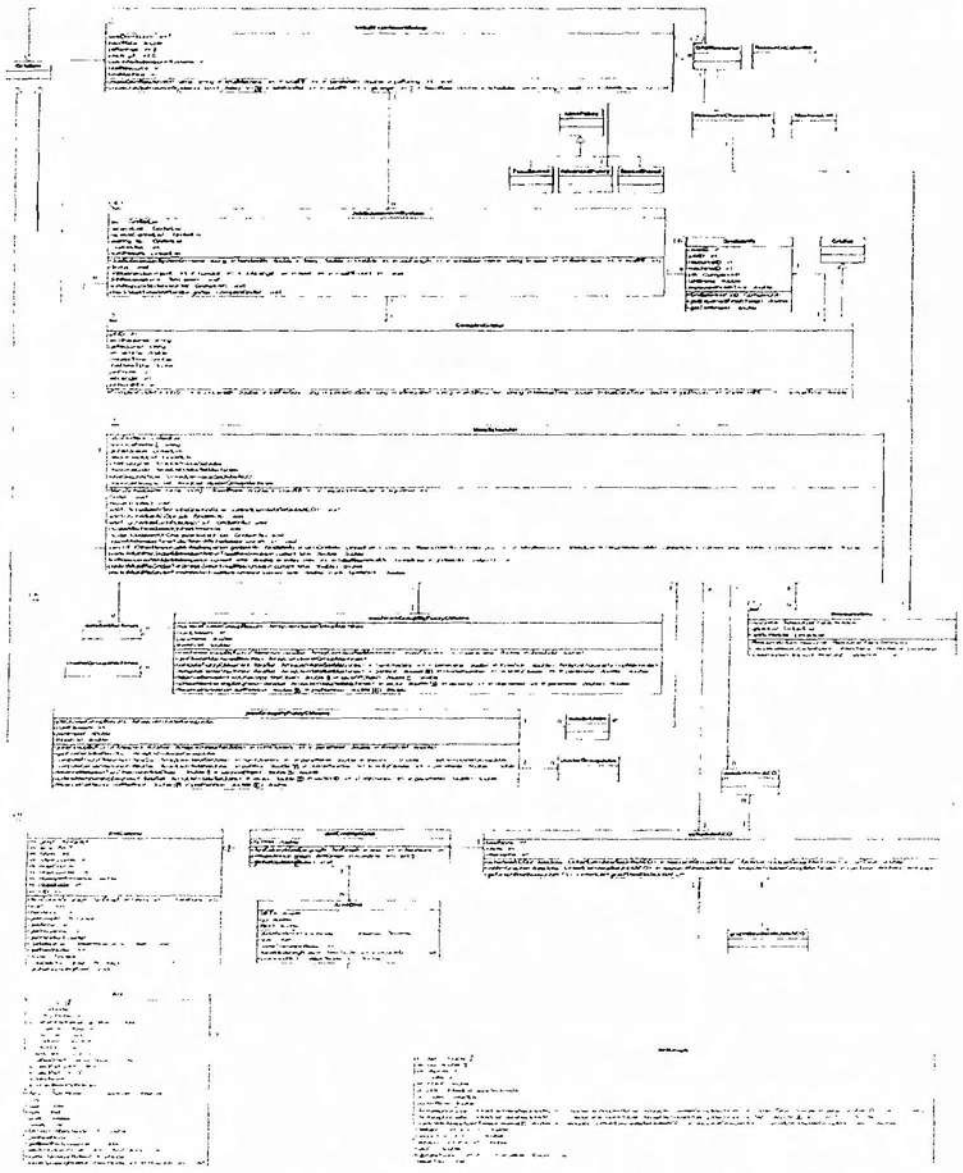

Fig 3. A class diagram showing the relationship in UML among GridSim, MetaScheduler and other entities

4. EXPERIMENT AND RESULTS

The experimental simulation is designed and implemented based on Grid simulation architecture described in section 3. The experiment is performed on Intel Pentium 4 2.6GHz machine with 640 MB RAM. The experimentation is differently setup of the computation workload of the job and it is shown in Table I, and the different resource is shown in Table II.

Table I: Jobs workload attributes

Parameters	Values	Notations
Total number of job	3,000	
Length of a job	1,000 – 5,000	MI
Arrival time distribution	0 – 20,000	Uniform distribution
Number of CPUs requirement	1-4 CPUs	Processing Elements (PEs)

Table II: The Grid resources attributed

Parameters	Values	Notations
Total number of resource	10 - 20	machines
Speed of CPU	50 - 200	MIPS
bandwidth	10	Mbit
Number of CPUs each machine	1-4 CPUs	Processing Elements (PEs)

In Fig 4, had shown the results of comparison between FCFS, MCT, Min-Min and ACO algorithms, the results are clear that the tardiness time based on the number of available machines in the Grid system. Although the ACO algorithm is good when was compared with the basic algorithms but it still had not good enough when compared with MCT and Min-Min dispatching rules that could be able to perform the sub-optimal scheduling job in value of the objective function, where the MCT and Min-Min accounted for less than 63%of the total make-span time in the average, when compared with ACO. ACO performed whenever every ten new jobs arrived to the system. On the other hand, the results had been shown in Fig 5, an FCFS algorithm performed much faster than the other scheduler algorithms because it basically performed the calculation before the job was assigned to a resource, as, the calculation time of MCT, Min-Min and ACO were much consumed the resource. In Fig. 6, had shown the percentage of the utilization in Grid environment, the best value was ACO. As, FCFS lower value than the other algorithms. Fig. 7, had show total tardiness time. Make-span time reflects demands of administrator on maximizing resource usage and throughput of all the system. On the other hand, total tardiness focuses on the demand of Grid user.

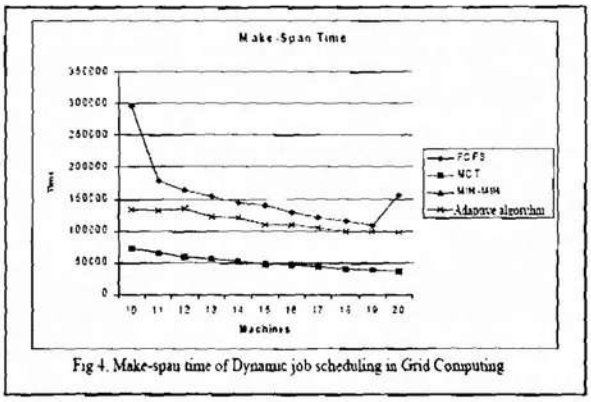

Fig 4. Make-span time of Dynamic job scheduling in Grid Computing

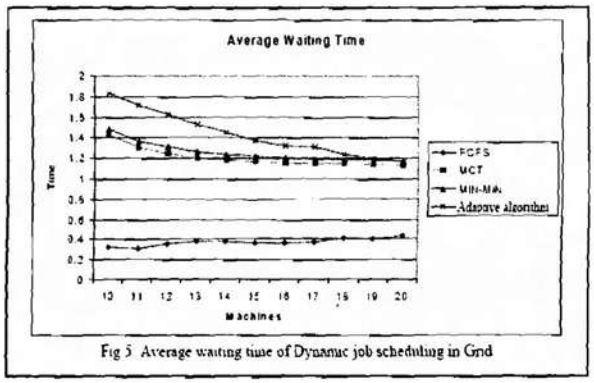

Fig 5 Average waiting time of Dynamic job scheduling in Grid

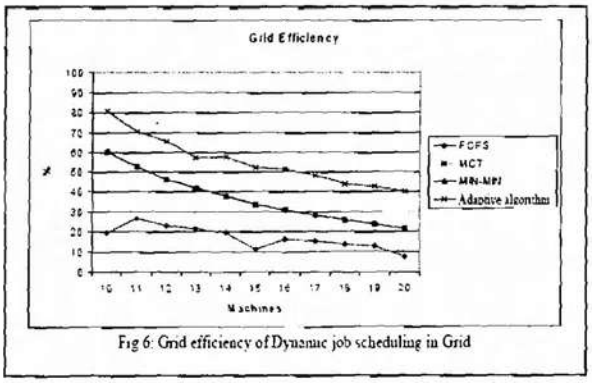

Fig 6: Grid efficiency of Dynamic job scheduling in Grid

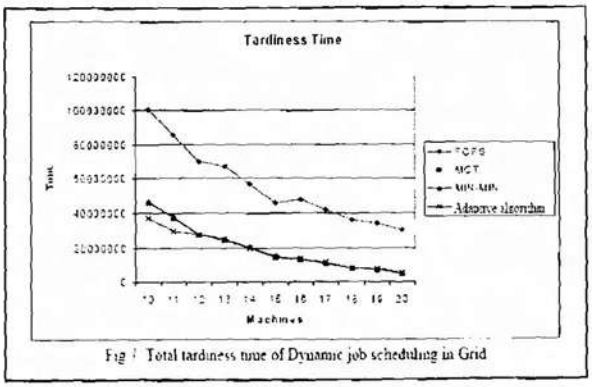

Fig 7 Total tardiness time of Dynamic job scheduling in Grid

5. CONCLUSION

This paper investigates the job online scheduling algorithm in grid environments like as an optimization problem. The solution is developed based on an Adaptive intelligent scheduling system to find a proper resource allocation on each processing job. Moreover, the popular GridSim toolkit for grid simulation is implemented and an Adaptive intelligent scheduling system is capable to play with the simulation.

In the study, the algorithm is designed within dynamic CPUs load, in which the natural grid usually occur different load of CPUs from time to time. To addition, this paper makes effort to design probably adaptive algorithm based on multiple objective of different stakeholder, for example makespan time, average waiting time and grid efficiency

6. FUTURE WORK

The future work will show the comparison with tradition algorithm FCFS for example and another heuristics such as MCT, Climbing Search, Simulated Annealing and Tabu search. Moreover, to show the evaluation of scheduling algorithms, in which this algorithm will probably get good enough results.

ACKNOWLEDGEMENTS

We would like to thank Suan Dusit Rajabhat University for supporting us.

REFERENCES

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure* 1st ed.: Morgan Kaufmann, 1999.

[2] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Transactions on Software Engineering,* vol. 15, p. 1427, 1989.

[3] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing," in *Grid Computing — GRID 2000*, 2000, pp. 191-202.

[4] G. Ritchie and J. Levine, "A Fast, Effective Local Search for Scheduling Independent Jobs in Heterogeneous Computing Environments," in *PLANSIG 2003: Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, Glasgow, 2003.

[5] D. B. Tracy, S. Howard Jay, B. Noah, L. B. Lasislau, l, ni, M. Muthucumara, I. R. Albert, P. R. James, D. T. Mitchell, Y. Bin, H. Debra, and F. F. Richard, "A comparison of eleven

static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, pp. 810-837, 2001.

[6] R. Buyya and M. Manzur, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, pp. 1175-1220, 2002.

[7] D. Klus´a˘cek, L. Matyska, and H. Rudov´a, "Alea - Grid Scheduling Simulation Environment," *Lecture Notes in Computer Science*, 2007.

[8] G. F. Dror, R. Larry, S. Uwe, C. S. Kenneth, and W. Parkson, "Theory and Practice in Parallel Job Scheduling," in *Proceedings of the Job Scheduling Strategies for Parallel Processing*: Springer-Verlag, 1997.

[9] D. Feitelson, "Packing schemes for gang scheduling," *Job Scheduling Strategies for Parallel Processing*, vol. Volume 1162/1996, pp. 89-110, 1996.

[10] K. Jochen, S. Uwe, and Y. Ramin, "On the Design and Evaluation of Job Scheduling Algorithms," in *Proceedings of the Job Scheduling Strategies for Parallel Processing*: Springer-Verlag, 1999.

[11] N. Randolph, T. Don, and N. T. Asser, "Performance Analysis of Parallel Processing Systems," *IEEE Trans. Softw. Eng.*, vol. 14, pp. 532-540, 1988.

[12] J. van den Akker, J. Hoogeveen, and J. van Kempen, "Parallel Machine Scheduling Through Column Generation: Minimax Objective Functions," in *Algorithms – ESA 2006*, 2006, pp. 648-659.

[13] E. Carsten, H. Volker, and Y. Ramin, "Benefits of Global Grid Computing for Job Scheduling," in *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*: IEEE Computer Society, 2004.

[14] S. Hongzhang, O. Leonid, and B. Rupak, "Job Superscheduler Architecture and Performance in Computational Grid Environments," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*: IEEE Computer Society, 2003.

[15] L. Keqin, "Job scheduling and processor allocation for grid computing on metacomputers," *J. Parallel Distrib. Comput.*, vol. 65, pp. 1406-1418, 2005.

[16] S. Vijay, K. Rajkumar, S. Srividya, and P. Sadayappan, "Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*: IEEE Computer Society, 2002.

[17] "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Pract. Exper.*, vol. 32, pp. 135-164, 2002.

[18] S. Lorpunmanee, M. N. M. Sap, A. H. Abdullah, and S. Surat, "Meta-scheduler in Grid environment with multiple objectives by using genetic algorithm," *WSEAS TRANSACTIONS on COMPUTERS* vol. 5, pp. 484 - 491, 2006.

[19] C. Henri, Z. Dmitrii, B. Francine, and L. Arnaud, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," in *Proceedings of the 9th Heterogeneous Computing Workshop*: IEEE Computer Society, 2000.

[20] B. Ranieri, F. Renato, and R. Pierluigi, "A static mapping heuristics to map parallel applications to heterogeneous computing systems: Research Articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, pp. 1579-1605, 2005.

[21] P. Fibich, L. Matyska, and H. Rudov´a, "Model of Grid Scheduling Problem," in *In Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing* 2005.

[22] X. Zhihong, H. Xiangdan, and S. Jizhou, "Ant algorithm-based task scheduling in grid computing," in *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, 2003, pp. 1107-1110 vol.2.

[23] E. Lu, Z. Xu, and J. Sun, "An Extendable Grid Simulation Environment Based on GridSim," in *Grid and Cooperative Computing*, 2004, pp. 205-208.

[24] H. Yan, X. Shen, X. Li, and M. Wu, "AN IMPROVED ANT ALGORITHM FOR JOB SCHEDULING IN GRID COMPUTING," in *In Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, Guangzhou, 2005.

[25] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, second ed.: Prentice Hall, 1995.

[26] "http://www.globus.org/toolkit/."

[27] A. Pugliese, D. Talia, and R. Yahyapour, "Modeling and Supporting Grid Scheduling," *Journal of Grid Computing*.

[28] F. W. Glover and M. Laguna, *Tabu Search* 1ed.: Kluwer Academic, 1997.

[29] "A Tutorial on Clustering Algorithms: Fuzzy C-Means Clustering, http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/index.html."