سومین کنفرانس فناوری اطلاعات و دانش
۶ تا ۸ آذرماه ۱۳۸۶

Distributed Resource Scheduling in Grid Computing Using Fuzzy Approach

Hamed Vahdat-Nejad, Reza Monsefi, Hossein Deldari

Computer Engineering Department, Ferdowsi University of Mashhad, Iran
{Hamed.vh@gmail.com, rmonsefi@um.ac.ir, hdeldari@yahoo.com }

Abstract
Scheduling is a fundamental issue in achieving high performance in multiclusters and computational grids. In this paper, we investigate the scheduling problem using the multilayer scheduling model. For attaining scalability, the proposed scheduler uses a distributed approach with the capability of considering the local clusters advantages for executing the jobs. On the other hand, the distributed scheduler exploits the capabilities of fuzzy logic to qualitatively deal with different parameters available in the scheduling decision. Simulation results show the effectiveness of the algorithm in terms of job completion time.

Keywords: Fuzzy theory, Grid computing, Job scheduling, Distributed scheduling

1. Introduction

A grid computing infrastructure is a collection of resources connected by a network, in which, by means of appropriate software, resource discovery and sharing is made possible [1]. One way to create a platform for grid computing is to interconnect existing separate clusters. These clusters may be located within a single organization or across different administrative organizations [2]. Scheduling is an important issue in grid computing, and parallel jobs constitute a typical workload in the scheduling scenario. Parallel jobs can be classified into two categories: Rigid and moldable [2]. Rigid parallel jobs run on a user specified number of resources, while moldable jobs can run on different number of computational resources. In this paper we consider rigid parallel jobs.

One class of applications that typically run on a grid superstructure is the class of single-program-multiple-data (SPMD) applications, also known as data-parallel applications. Data-parallel applications are partitioned into tasks which perform computations on separate pieces of a data set. The tasks work together to process the entire data set, and are collectively called a job. The data-parallel program model is often used to solve scientific computing problems. These jobs may run for many hours or days, and can consume a large amount of system resources. The job may perform a large amount of computation, communication between tasks, or both [1].

One category of rigid jobs that we have investigated is data parallel applications. A grid scheduler uses the information of grid system and jobs to produce an assignment of tasks to machines for the given grid job. The general problem of mapping tasks to machines has been shown to be NP-complete [3]. The scheduling of parallel jobs has been extensively studied in a single cluster environment [5, 6]. Several heuristic algorithms have been developed to schedule tasks to machines on heterogeneous computing systems. Eleven such scheduling algorithms have been evaluated in [4]. These algorithms are developed for heterogeneous computing systems. Some heuristic scheduling algorithms for grid environments are developed in [7, 8, 9]. They deal with tasks and machines in terms of assigning tasks to machines, and have the deficiency of not being scalable, when applied to a large scale grid. For attaining scalability, we use a distributed approach, in a way that the arrival job can be submitted to any of the

clusters. Afterward a 2 layer (global and local) scheduling scheme is deployed, which its first stage is responsible for assigning the job to an appropriate cluster. This stage is called global scheduling, which in fact schedules the job at the grid-level. Afterward the cluster's scheduler submits the job to the scheduler of the selected cluster, which in turn starts the scheduling of job's tasks in its local nodes upon receiving the job. The local scheduler (cluster-level scheduler) uses the first-in first-out policy for scheduling.

In this study we focus on distributed global scheduling (scheduling at the grid-level), which deals with jobs and clusters, and assigns each job to a cluster. For this, we consider the computational needs (i.e. the number of computational resources on which the job is requested to be run), and communication requirements (the amount of communication between job's tasks) of the job. We believe that job's communication ratio is an important parameter in scheduling. Existing schedulers, presently, ignore issues such as network load and the communication requirements of the application. In a typical scenario, if an incoming job requires a fixed number of computational resources and the same number of machines is available in a cluster, the job is submitted to that cluster, even if the load on the cluster network precludes the completion of the job by an acceptable time. The work presented in this paper, addresses the need to build a fuzzy system that augments the scheduling capabilities by incorporating issues such as cluster's network load and job's communication requirements. The specific question being addressed in this paper is which of the clusters should be allocated to an incoming job. This question requires a process of prioritization of the available clusters. For this, fuzzy logic was used in the form of a controller that contains rules which match the resource specification requirements of the jobs to the available resources of the clusters. The objective is to include cluster's network load as a cluster resource, and job's communication requirements as a resource specification in the prioritization process. Using fuzzy logic, it is possible to reason about these parameters in a qualitative manner, and at the same time to improve the scheduling decisions.

The Rest of the paper is organized as follows. In section 2, we describe the scheduling model, in which the assumptions about the grid and incoming jobs are expressed. In section 3, the proposed global scheduling algorithm is presented. In section 4, we evaluate the global scheduler through simulation, and section 5 concludes the paper.

2. The scheduling model and assumptions

The assumed multicluster consists of m clusters $C_1,C_2\ldots,C_m$, and each cluster is composed of a number of homogenous computational resources, and a scheduler. The scheduling is done in two levels: global and local. The arrival job could be submitted to the scheduler of any of the clusters. The cluster which the job is submitted to is called local cluster, and the others are called remote. The scheduler of the local cluster should decide where the arrival job is going to be run. It may choose the local or one of the remote clusters for executing the job. This decision is done through global scheduling, which is the main subject of this paper. If the selected cluster is one of the remote ones, the local scheduler submits the whole job to the scheduler of that cluster. After this stage, the scheduler of the cluster which finally receives the job is responsible for scheduling the parallel job within its local nodes. This stage of the scheduling process is called local scheduling, which has previously been the subject of some papers [5,6]. Figure 1 shows an overall schema of the multicluster under investigation.

Parallel jobs considered in this paper are rigid. The job model is built from user-provided application characteristics that do not require extensive job profiling. They are

- The number of partitioned tasks.
- The ratio of communication to execution.

The ratio of communication to execution gives a method of weighing the relative importance of communication rates and computational power for the job, without requiring extensive application profiling.

In summary, a parallel job, denoted by J_i, is identified by a 3-tuple (A_i , N_i , R_i), where A_i is J_i's arrival time, N_i is the number of computational resources on which J_i requested to be run, R_i is the ratio of communication to execution.

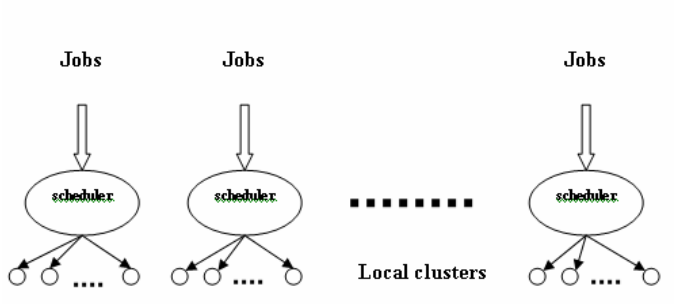

Figure 1: The multicluster superstructure

3. scheduling

After a user submits a job to one of the clusters, the global scheduling process is started by the scheduler of that cluster. The goal is to find a suitable cluster for assigning the job to it. The global scheduling consists of two stages: In the first stage, all of the clusters are considered equally, and a priority is assigned to each of them. Afterwards the cluster with the highest priority is selected as a candidate for assigning the Job.

When the ultimate cluster for assigning the job is a remote cluster, a requirement of permission is arisen, because one or more than one job may have been scheduled to that cluster by its local scheduler or other schedulers during the interval between choosing that cluster for scheduling and submitting the job. Hence before submitting a job to a remote cluster, the local scheduler should send a permission message to that cluster, and upon receiving the "ok" response, it submits the job. In reverse, if the response is "No", the local scheduler repeats the global scheduling

process after updating the state information of other clusters.

3.1 The proposed global scheduling algorithm

For the characterization of the state of the cluster, we consider two parameters: the available number of CPUs and the cluster's network load. What is meant by cluster's network load is the level of communication presently trafficking through the cluster, which is a ratio between zero and one. When a job arrives, the scheduler is triggered to assign the job to a cluster. For this, it considers all the clusters and assigns two weights (which are numbers between zero and one) to each cluster. The first weight (w_1), also called cluster weight for number of machines, determines a matching degree between the number of available low load CPUs in the cluster, and the number of tasks of the job. The second weight (w_2), also called cluster weight for network load, considers job's communication requirements, and cluster's network load. We have performed many simulation tests to adjust the coefficient of these weights in aggregating priority equation. The final formula obtained for calculating priority of a cluster with respect to the newly arrives job is given below

$$\text{Priority} = 0.7\ W_1 + 0.3\ W_2$$

After computing the priority of all the clusters, the scheduler assigns the job to the cluster with the maximum priority.

3.1.1 Cluster's weight for number of machines

Load of cluster's nodes is a dynamic attribute, and is computed by averaging the currently reported loads (CPU usage) of the node. The scheduler partitions the nodes of a cluster into low-load, medium-load, and high-load nodes. A node which its load is less than 0.3 is low-load; a node with load between 0.3 and 0.6 is medium-load; and a node with load higher than 0.6 is a high-load node. For the i^{th} cluster, L_i, M_i, H_i show the number of low-load, medium-load and high-load nodes, respectively. The following pseudo code shows the algorithm for computing the cluster's weight for number of machines (W_1).

---

if (L_i > numofTask)
{

$$W_1 = 1 - \left(\frac{L_i - numofTask}{L_i} \right)^3$$

If W_1<0.5 W_1=0.5;
}
else if ($L_i + M_i$ > numofTask)
{

$$W_1 = 0.5 - \left(\frac{numofTask - L_i}{numofTask} \right)^3$$

If W_1<0.2 W_1=0.2;
}
else if ($L_i + M_i + H_i$ >numofTask)
{

$$W_1 = 0.2 - \left(\frac{numofTask - (L_i + M_i)}{numofTask} \right)^3$$

If W_1<0 W_1=0;
}

Here, numofTask determines the number of partitioned tasks of the job.

3.1.2 Cluster's weight for network load

One of the main features of this work is considering jobs' communication requirements and clusters network load. When a job has a high communication ratio, it must be scheduled to a cluster with low network load. For this, we use fuzzy logic to assign a weight to the cluster, which determines suitability of executing the job on the cluster. This assignment considers communication requirements of the job and network load of the cluster. The fuzzy rule based system has two input parameters: Jobs communication ratio, and available bandwidth of the cluster (as a ratio between zero and one), and one output: cluster's weight for network load (W_2). When job's communication ratio is close to one, it means that, the job requires high communication, so a small weight (close to zero) should be assigned to a cluster with a little (close to zero) available bandwidth, and a high weight (close to one) should be assigned to a cluster with high (close to one) available bandwidth. Figures 2 through 4 show examples of the fuzzy membership functions for these two inputs, and for the output being the weight assigned to network load of the cluster. Table I summarizes the rules that map the inputs to the output. We have used product inference engine, singleton fuzzifire, and center average defuzzifire.

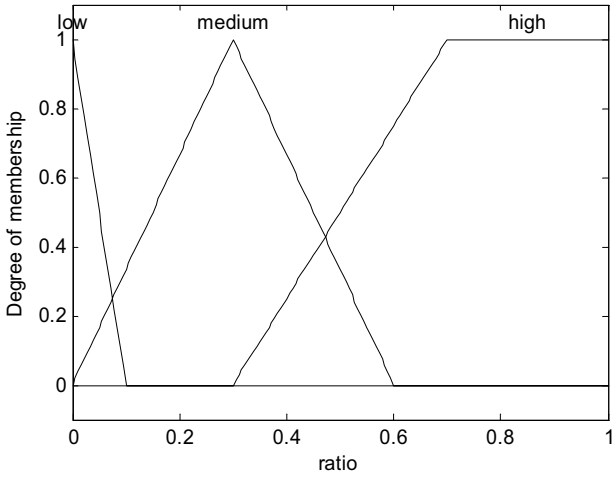

Figure 2: Fuzzy membership functions of communication to execution ratio

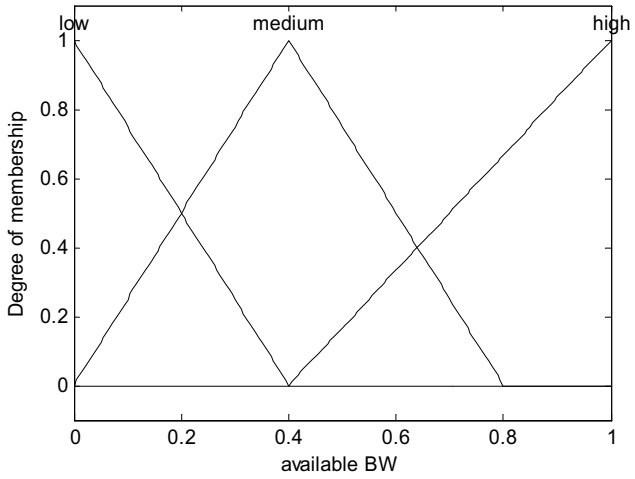

Figure 3: Fuzzy membership functions of available BW

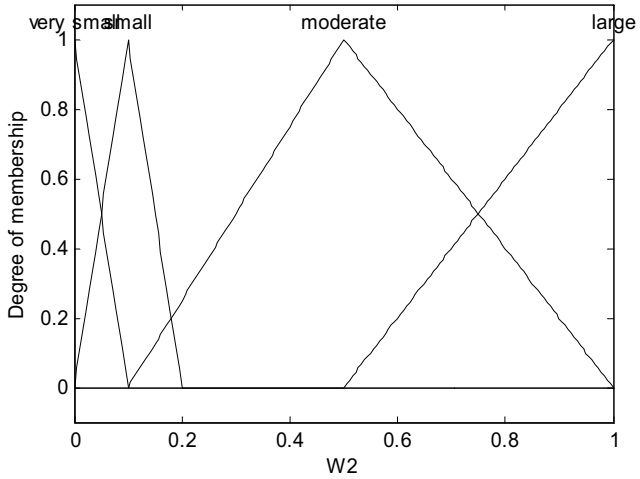

Figure 4: Fuzzy membership functions of cluster weight

Table I. Fuzzy rules for mapping inputs to the output

Ratio/Available BW	Low	Medium	High
Low	Moderate	Large	Large
Medium	Small	Moderate	Large
High	Very small	Small	Large

4. Simulation results

We have developed a simulator in Matlab to evaluate the performance of the proposed scheduling algorithm. The simulated multicluster consists of several clusters, each of which composed of 1 to 100 nodes. Interarrival time of jobs is considered to satisfy a Poisson process with the parameter λ seconds. Each job is submitted with three attributes: arrival time, number of tasks, and communication to execution ratio. Each job has a random number of tasks between 1 and 50, and a random communication to execution ratio between 0 and 0.7. The scheduler does not need to predict the job's execution time.

We compare our scheduling algorithm with a distributed best-fit policy, which ignores the communication requirements of the jobs, and available bandwidth of the clusters. The best-fit scheduling algorithm assigns the submitted job to a cluster whose number of idle nodes is greater than the number of tasks of the job, and whose number of idle nodes is the least . There are two scheduling scenarios in the simulation.

Scenario I: 5 clusters, 15 jobs, $\lambda = 12$.

Scenario II: 30 clusters, 50 jobs و $\lambda = 16$.

For each scenario systems are automatically generated. Figure 6 shows parallel jobs completion time for scenario I. Small rectangles and parallelograms show completion time of jobs in the best-fit and the proposed algorithm, respectively. In Figure 7 parallel jobs completion time are shown for scenario II. In both figures, horizontal axis shows the arrival time of jobs, and vertical axis represents the completion time of the submitted jobs. The completion time of a job is computed as the interval between job's arrival time and job's completion time. As can be seen in both Figures 6 and 7, we conclude that the proposed fuzzy algorithm is better than the best-fit algorithm in reducing parallel completion time. The proposed algorithm uses the job's communication requirements, and cluster's network load to take scheduling decisions more efficiently, and as a result, it improves job's completion time.

.

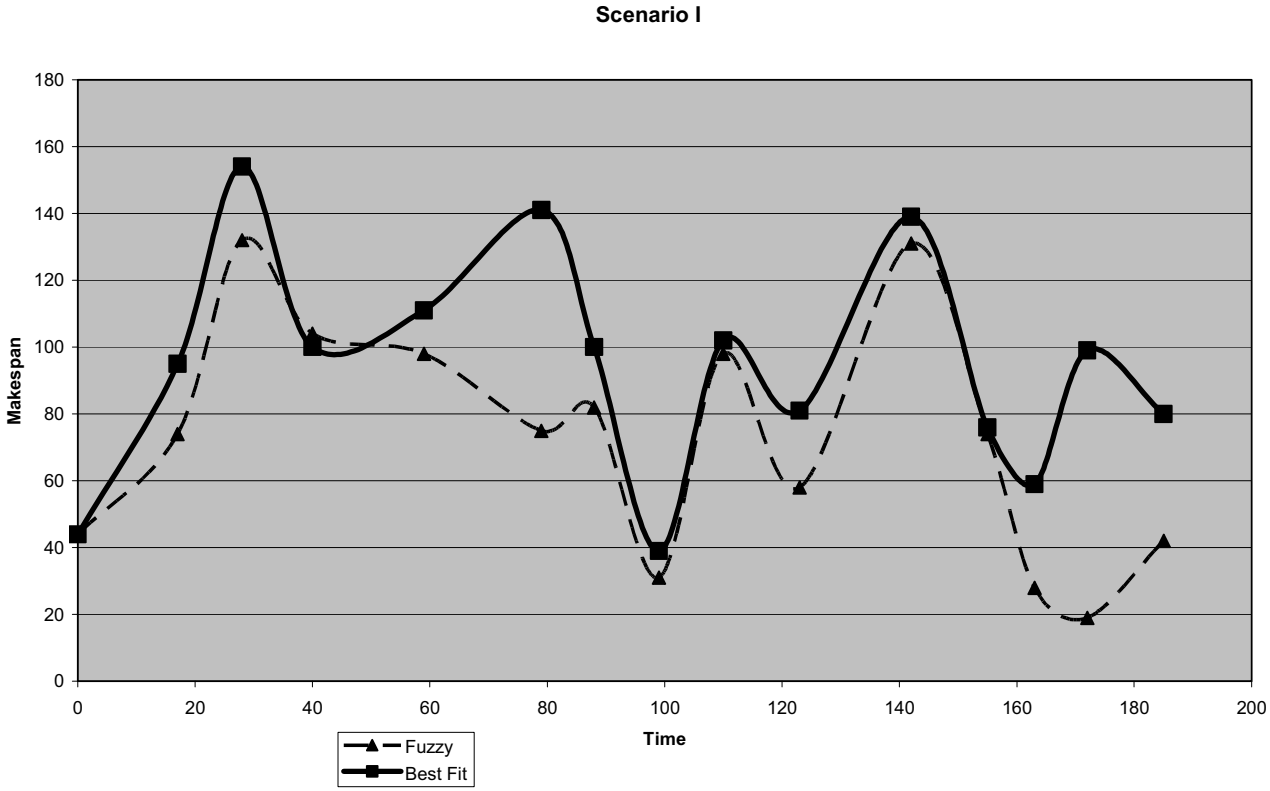

Figure 6: Simulation for parallel jobs completion time (Scenario I)

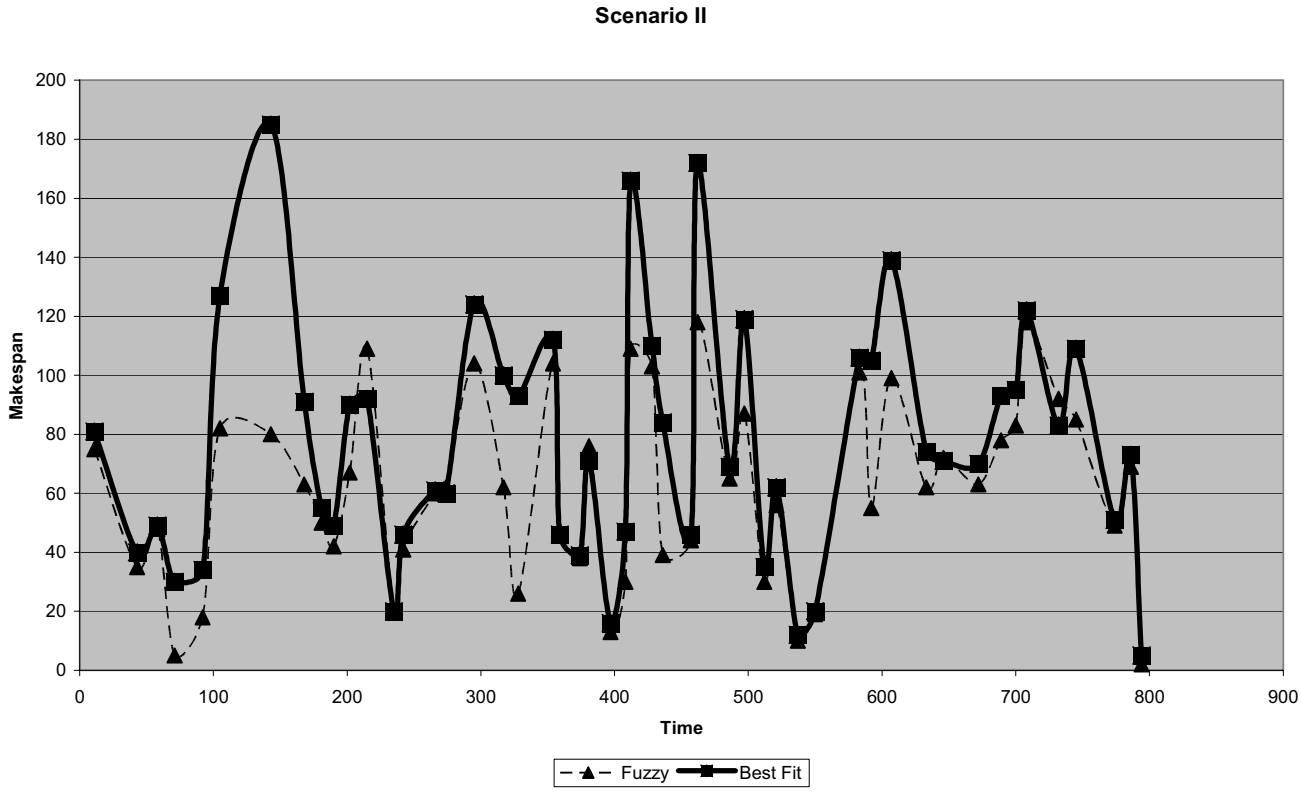

Figure 7: Simulation for parallel jobs completion time (Scenario II)

5. Conclusion

Job scheduling is very complicated in computational grids. Parallel jobs are a set of important applications that usually constitute the workflow of a grid. In this paper, we present a distributed scheduling algorithm for scheduling parallel jobs in a computational grid. The scheduling is done in two layers: global and local. We focus our work to global scheduling, which is responsible for allocating the submitted job to a cluster. The global scheduler assigns a priority, based on two matching degrees, to each cluster for a submitted job. It then allocates the cluster with highest priority to the job. First matching degree (w_1) expresses the suitability of executing the job on the cluster in terms of the number of tasks of the job, and the number of available nodes in the cluster. The other (w_2) expresses the suitability in terms of cluster's available bandwidth, and job's communication requirements. For computing w_2, we use a fuzzy rule based system to consider different parameters in a qualitative manner. The simulation results show the improvement of the proposed algorithm over a distributed best-fit policy which ignores communication requirements of the jobs and network traffic of the clusters.

References

[1] Michael Walker, A Framework for Effective Scheduling of Data-Parallel Applications in Grid Systems, Master thesis, University of Virginia, 2001.

[2] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, Xinuo Chen and Graham R. Nudd, Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids, Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004

[3] HU Rong, HU Zhigang, A Scheduling Algorithm Aimed at Time and Cost for Meta-tasks in Grid Computing Using Fuzzy Applicability, Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region, IEEE 2005.

[4] Tracy D Braun, Howard Jay Siegel et al, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing system, Journal of Parallel and Distributed computing, 2001, 6, pp.810-837.

[5] B. G. Lawson and E. Smirni, Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems, Proceedings of the Eighth Job Scheduling Strategies for Parallel Processing, 2002.

[6] E. Shmueli and D. G. Feitelson, Backfilling with look ahead to optimize the performance of parallel job scheduling, in Job Scheduling Strategies for Parallel Processing, D. G. Feitelson, L.Rudolph, and U. Schwiegelshohn (Eds.), pp.228-251, Springer-Verlag, 2003.

[7] H. Yan, et, al, An Improved Ant Algorithm for Job Scheduling in Grid Computing, Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005.

[8] Y. Hu, et. Al, An Algorithm for Job Scheduling in Computational Grid Based on Time-Balancing Strategy, Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005.

[9] Liang-Teh Lee Chin-Hsiian Liang Hung-Yuan Chang, An Adaptive Task Scheduling System for Grid Computing, Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, 2006.