Future Generation Computer Systems 25 (2009) 20–27

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

An ant algorithm for balanced job scheduling in grids

Ruay-Shiung Chang *, Jih-Sheng Chang, Po-Sheng Lin

Department of Computer Science and Information Engineering, National Dong Hwa University, Shoufeng Hualien, 974 Taiwan, ROC

ARTICLE INFO

Article history:
Received 24 August 2007
Received in revised form
17 June 2008
Accepted 17 June 2008
Available online 29 June 2008

Keywords:
Ant algorithm
Job scheduling
Grid computing

ABSTRACT

Grid computing utilizes the distributed heterogeneous resources in order to support complicated computing problems. Grid can be classified into two types: computing grid and data grid. Job scheduling in computing grid is a very important problem. To utilize grids efficiently, we need a good job scheduling algorithm to assign jobs to resources in grids.

In the natural environment, the ants have a tremendous ability to team up to find an optimal path to food resources. An ant algorithm simulates the behavior of ants. In this paper, we propose a Balanced Ant Colony Optimization (BACO) algorithm for job scheduling in the Grid environment. The main contributions of our work are to balance the entire system load while trying to minimize the makespan of a given set of jobs. Compared with the other job scheduling algorithms, BACO can outperform them according to the experimental results.

1. Introduction

Current scientific problems are very complex and need huge computing power and storage space. The past technologies such as distributed or parallel computing are unsuitable for current scientific problems with large amounts of data. Processing and storing massive volumes of data may take a very long time. Grid computing [1] is a new paradigm for solving those complex problems. In grids, we need to consider the conditions such as network status and resources status. If the network or resources are unstable, jobs would be failed or the total computation time would be very large. So we need an efficient job scheduling algorithm for these problems in the grid environment.

The purpose of job scheduling is to balance the entire system load while completing all the jobs at hand as soon as possible according to the environment status. Because the environment status may change frequently, traditional job scheduling algorithm such as "First Come First Serve" (FCFS), "Shortest Job First" (SJF), etc., may not be suitable for the dynamic environment in grids.

In grids, users may face hundreds of thousands of computers to utilize. It is impossible for anyone to manually assign jobs to computing resources in grids. Therefore, grid job scheduling is a very important issue in grid computing. For example, in BOINC [2], an open-source software for volunteer computing and grid computing, job scheduling is one of the most important key factors for achieving Teraflops performance [3]. Because its importance, many job scheduling algorithms for grids [4–6] have been proposed. Please refer to a survey [7], which also poses some open issues.

A good schedule would adjust its scheduling strategy according to the changing status of the entire environment and the types of jobs. Therefore, a dynamic algorithm in job scheduling such as Ant Colony Optimization (ACO) [8,9] is appropriate for grids.

ACO is a heuristic algorithm with efficient local search for combinatorial problems. ACO imitates the behavior of real ant colonies in nature to search for food and to connect to each other by pheromone laid on paths traveled. Many researches use ACO to solve NP-hard problems such as traveling salesman problem [10], graph coloring problem [11], vehicle routing problem [12], and so on.

This paper applies the ACO algorithm to job schedule problems in Grid computing. We assume each job is an ant and the algorithm sends the ants to search for resources. We also modify the global and local pheromone update functions in ACO algorithm in order to balance the load for each grid resource. Finally, we compare the proposed BACO (Balanced ACO) algorithm with iACO (Improved ACO) [13], FPLTF (Fastest Processor to Largest Task First) [14], dynamic FPLTF [15], Sufferage [15], and random selection method in the experiments. According to the experimental results, we can find out that BACO is capable of achieving system load balance better than other job scheduling algorithms.

The rest of the paper is organized as follows. Section 2 introduces the related work about many kinds of ACO algorithm and job scheduling in grids. Section 3 details the proposed ACO algorithm in job scheduling. Section 4 is the experimental results. Finally, Section 5 concludes this paper.

* Corresponding author.
E-mail address: rschang@mail.ndhu.edu.tw (R.-S. Chang).

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.06.004

2. Related work

2.1. Ant algorithms

There are many different kinds of ACO algorithm, i.e., Ant Colony System (ACS) [10], Max-Min Ant System (MMAS) [16], Rank-based Ant System (RAS) [17], Fast Ant System (FANT) [18] and Elitist Ant System (EAS) [19]. ACS uses the *pseudo-random-proportional rule* to replace state transition rule for decreasing computation time of selecting paths and update the pheromone on the optimal path only. It is proved that it helps ants search the optimal path.

MMAS is based on the basic ACO algorithm but limiting the pheromone range to be greater than or equal to the low bound value (Min) and smaller than or equal to the upper bound value (Max). The low bound and upper bound are defined by the user. According to the low bound and upper bound values, MMAS could avoid ants to converge too soon in some ranges.

In the design of RAS, it sorts the ants by ant's tour length in ascending order after all ants completed their tours. It means that the first ant finds the shortest path to complete the tour and the last ant takes the longest tour. They give each ant a different density of pheromone to update their path by the ascending order: the higher the position of the ant, the more pheromone it could update; the lower the position of the ant, the less pheromone it has. By the idea of RAS, the shortest length gets more pheromone to attract more ants to follow and the system could get the optimal solution very soon.

FANT employs one ant at each iteration and uses the solution of the ant to do a local search. FANT works without evaporation rule and it updates pheromone after each iteration. In order to avoid the sub-optimal solution, it applies a reset pheromone function.

EAS update more pheromone on the best-so-far tour found in order to attract more ants to follow the best-so-far tour.

There are many studies about job scheduling using ACO algorithm in grid environment such as [13]. It uses the basic idea of ACO, but changes the pheromone update rule by adding encouragement, punishment coefficient and load balancing factor.

In [20], Kwang Mong Sim et al. use multiple kinds of ant to find multiple optimal paths for network routing. The idea can be applied to find multiple available resources to balance resources utilization in job scheduling. The key of the idea is each different kinds of ant can only sense their own kind of pheromone so that it can find many different paths including the shortest-path by different kinds of ant. There are still some problems that if all kinds of ant find the same path, it will be the same as using one kind of ant. How to compare the performance for each kind of ant creates another problem. Furthermore, one solution from this algorithm may work efficiently in an environment, but it may work inefficiently in another one.

In [21], J. Heinonen et al. apply the hybrid ACO algorithm with different visibility to job-shop scheduling problem. The hybrid ACO algorithm consists of two ideas. One idea is the basic ACO algorithm, and the other idea uses the post-processing algorithm in the part of local search in ACO algorithm. When the ACO algorithm finished, all ants complete its own tours. A tour can be decomposed into blocks. The block for swap must contain more than two operations. Then the post-processing algorithm uses the swap operation on the blocks. If the swap refines the makespan, the new path is accepted; otherwise the swap is invalid and the swapped block reverts to previous status.

The ACO algorithm has also been applied to hard combinatorial optimization problems such as traveling salesman problem (TSP) [10], flow shop problem [22], project presentation scheduling [23], graph coloring problem [11], vehicle routing problem [12], and nurse scheduling [24], and so on.

2.2. Job scheduling in grids

Job scheduling is well studied within the computer operating systems [25]. Most of them can be applied to the grid environment with suitable modifications. In the following we introduce several methods for grids.

The FPLTF (Fastest Processor to Largest Task First) [14] algorithm schedules tasks to resources according to the workload of tasks in the grid system. The algorithm needs two main parameters such as the CPU speed of resources and workload of tasks. The scheduler sorts the tasks and resources by their workload and CPU speed then assigns the largest task to the fastest available resource. If there are many tasks with heavy workload, its performance may be very bad. Dynamic FPLTF (DPLTF) [15] is based on the static FPLTF, it gives the highest priority to the largest task. DPLTF needs prediction information on processor speeds and task workload.

The WQR (Work Queue with Replication) is based on the work queue (WQ) algorithm [15]. The WQR sets a faster processor with more tasks than a slower processor and it applies FCFS and random transfer to assign resources. WQR replicates tasks in order to transfer to available resources. The amount of replications is defined by the user. When one of the replication tasks is finished, the scheduler will cancel the remaining replication tasks. The WQR's shortcoming is that it takes too much time to execute and transfer replication tasks to resource for execution.

Min-min [26] set the tasks which can be completed earliest with the highest priority. The main idea of Min-min is that it assigns tasks to resources which can execute tasks the fastest. Max-min [26] set the tasks which has the maximum earliest completion time with the highest priority. The main idea of Max-min is that it overlaps the tasks with long running time with the tasks with short running time.

For instance, if there is only one long task, Min-min will execute short tasks in parallel and then execute long task. Max-min will execute short tasks and long task in parallel.

The RR (Round Robin) algorithm focuses on the fairness problem. RR uses the ring as its queue to store jobs. Each job in queue has the same execution time and it will be executed in turn. If a job can't be completed during its turn, it will store back to the queue waiting for the next turn. The advantage of RR algorithm is that each job will be executed in turn and they don't have to wait for the previous one to complete. But if the load is heavy, RR will take long time to complete all jobs.

Priority scheduling algorithm gives each job a priority value and uses it to dispatch jobs. The priority value of each job depends on the job status such as the requirement of memory sizes, CPU time and so on. The main problem of this algorithm is that it may cause indefinite blocking or starvation if the requirement of a job is never being satisfied.

The FCFS (First Come First Serve) algorithm is a simple job scheduling algorithm. A job which makes the first requirement will be executed first. The main problem of FCFS is its convoy effect [25]. If all jobs are waiting for a big job to finish, the convoy effect occurs. The convoy effect may lead to longer average waiting time and lower resource utilization.

3. The balanced ant colony optimization (BACO) algorithm

BACO inherits the basic ideas from ACO algorithm to decrease the computation time of jobs executing in Taiwan UniGrid [27] environment and it also considers about the loading of each resource. BACO changes the pheromone density according to the resources status by applying the local pheromone update and the global pheromone update functions. The purpose is to try to minimize the completion time for each job while balancing the system load.

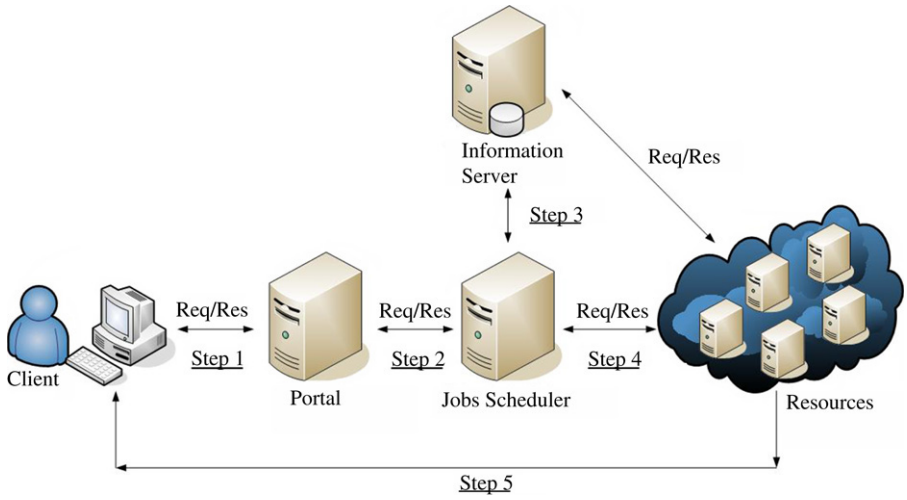

Fig. 1. System architecture.

3.1. System architecture

The system architecture is shown in Fig. 1. There are four main components: Portal, Information Server, Jobs Scheduler and grid resources. The Portal provides an interface to users for job execution. The Information Server collects resource information by using the NWS (Network Weather Service) [23]. The NWS demon reports system information back to Information Server periodically. The Jobs Scheduler selects the most appropriate resources to execute the request according to the proposed BACO algorithm. Finally, the execution results would be sent back to the user.

3.2. The proposed BACO algorithm

In order to map the ant system to the grid system, we explain their relationships below:

A. An ant

An ant in the ant system is a job in the grid system.

B. Pheromone

Pheromone value on a path in the ant system is a weight for a resource in the grid system. A resource with a larger weight value means that the resource has a better computing power.

The scheduler collects data from Information Server and uses the data to calculate a weight value of a resource.

The pheromone (weight) of each resource is stored in the scheduler and the scheduler uses it as the parameters for BACO algorithm. At last, the scheduler selects a resource by a scheduling algorithm and it sends jobs to the selected resource by the APIs of the Globus Toolkit [29].

Fig. 2 shows the mapping between the ant system and the grid system.

The initial pheromone value of each resource for each job is equal to the pheromone indicator. The pheromone indicator of each resource for each job is calculated by adding the estimated transmission time and execution time of a given job when assigned to this resource. The estimated transmission time can be easily determined by $\frac{M_j}{Bandwidth_i}$ where M_j is the size of a given job j and $bandwidth_i$ is the bandwidth available between the scheduler and the resource. However, the other parameter, the job execution time, is hard to predict. Depending on the type of programs, many methods [30–34] can be used to estimate the program execution time. With that, the pheromone indicator is defined by:

Pheromone indicator:

$$PI_{ij} = \left[\frac{M_j}{bandwidth_i} + \frac{T_j}{CPU\_speed_i \times (1 - load_i)}\right]^{-1} \quad (1)$$

where PI_{ij} is the pheromone indicator for job j assigned to resource i, M_j is the size of a given job j, T_j is the CPU time needed of job j, CPU_speed_i (CPU speed), $load_i$ (current load) $andbandwidth_i$ (bandwidth between the scheduler and the resource) are the status of resource i. The load, bandwidth and CPU speed can be obtained using the Network Weather Service (NWS) [28].

The pheromone indicator tells that when a job is assigned to a resource, we consider the resource status, the size of jobs, and the program execution time in order to select a suitable resource for execution. The larger the value of PI_{ij} is, the more efficient it is for resource i to execute this job j.

Assume there are m resources and n jobs. We have the PI matrix as follows:

$$PI = \begin{array}{c} \\ r_1 \\ r_2 \\ \cdot \\ \cdot \\ \cdot \\ r_m \end{array} \begin{array}{c} \begin{array}{cccc} j_1 & & & j_2 \end{array} \\ \begin{bmatrix} PI_{11} & PI_{12} & \dots & PI_{1n} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ PI_{m1} & PI_{m2} & \dots & PI_{mn} \end{bmatrix} \end{array}$$

In each iteration, we select the *largest entry* from the matrix. Please note if another job scheduling discipline is used before BACO, then BACO selects among the available jobs produced by the scheduling. Assuming PI_{ij} is selected, then job j is assigned to resource i for execution. After a job is assigned to a resource, we apply (1) to the resource selected for each unassigned jobs in the PI matrix. This is called the local (row) pheromone update.

The global pheromone update is to recalculate the entire PI matrix. It is performed when a job is completed. However, for the resource that just completes executing this job, after applying (1), the new value of pheromone in the row corresponding to the resource will have to be multiplied $(1-\rho_i)$ further, where $1 > \rho_i \geq 0$. ρ_i indicates the overhead incurred in resource i after completing job j. When a local scheduler is also used in a resource, ρ_i may be used to indicate the decrease of new job priority after a job from the same user is completed.

Global pheromone update reflects the changes of network condition and resource status after a job is completed. It incorporates the dynamic nature of the system into the scheduling algorithm such that a better decision can be made at the next turn.

For n jobs and m resources, assigning the first job needs to calculate the PI matrix of nm entries. For the second job, only $(n-1)m$ entries remained in the PI matrix. Therefore, the total number of matrix entries computed is $m \times \sum_{i=1}^{n} i = \frac{mn(n+1)}{2}$. Therefore, BACO has good scalability even if n or m grows very large.

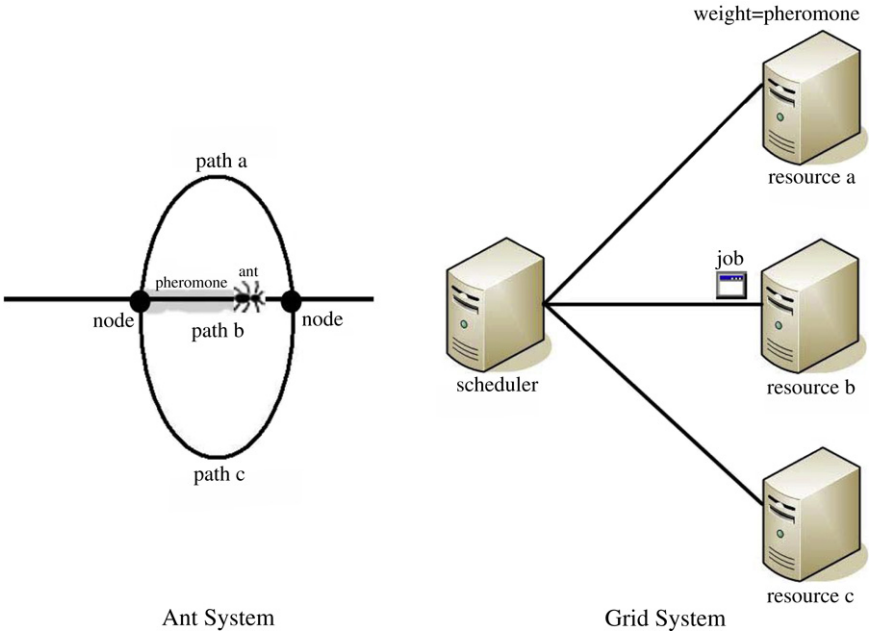

Fig. 2. Mapping between the ant system and the grid system.

Table 1
Initial status of each resource

	r_1	r_2	r_3
CPU speed (MHz)	3000	3200	2800
Load	25%	10%	30%
Bandwidth (Megabits/s)	11.69	25.20	15.95

3.3. Example of the proposed algorithm

Assume there are three jobs (j_1, j_2 and j_3) and three resources (r_1, r_2 and r_3) in the grid. Assume the initial status of each resource is shown in Table 1 and the size of each job is 3 MB, 2 MB and 1 MB. The CPU cycles needed for each job are 3 M, 2 M, and 1 M respectively. The initial pheromone indicator of each entry is shown in the following *PI* matrix.

$$PI = \begin{bmatrix} PI_{11} = 3.88 & PI_{12} = 5.82 & PI_{13} = 11.64 \\ PI_{21} = 8.33 & PI_{22} = 12.49 & PI_{23} = 24.99 \\ PI_{31} = 5.28 & PI_{32} = 7.91 & PI_{33} = 15.83 \end{bmatrix}.$$

At the beginning of job dispatch, the Job Scheduler selects the maximum pheromone indicator in the *PI* matrix, which is PI_{23}. So r_2 is selected for the execution of j_3. After assigning j_3 to r_2, we will update the second row of r_2 for all jobs by local update. Since j_3 is assigned, column 3 is no longer needed.

Assume the load of r_2 after the job assignment becomes 30% and the new *PI* matrix is as follows:

$$PI = \begin{bmatrix} PI_{11} = 3.88 & PI_{12} = 5.82 \\ PI_{21} = 5.83 & PI_{22} = 8.74 \\ PI_{31} = 5.28 & PI_{32} = 7.91 \end{bmatrix}$$

Local update

If r_2 finished j_3 before the Job Scheduler start to dispatch the next job, we will update all entry of *PI* matrix (global update) in order to get the newest pheromone for the next job submission. Assume the newest status of each resource after the execution of j_1 is shown in Table 2.

The ρ of r_1 and r_2 is zero because they have not been assigned jobs for execution and assume the ρ of r_2 is 0.05. The new PI matrix is as follows:

$$PI = \begin{bmatrix} PI_{11} = 3.47 & PI_{12} = 5.21 \\ PI_{21} = 6.53 & PI_{22} = 9.80 \\ PI_{31} = 4.61 & PI_{32} = 6.91 \end{bmatrix}$$

Table 2
The newest status of each resource after execution of j3

	r_1	r_2	r_3
CPU speed (MHz)	3000	3200	2800
Load	30%	20%	15%
Bandwidth (Megabits/s)	10.47	20.79	13.90
ρ	0	0.05	0

The remaining jobs are assigned similarly. When Job Scheduler assigns a job to a resource, the row of selected resource will be updated by the local pheromone function. After a resource finished a job, all entries of the *PI* matrix will be updated by the global pheromone update function.

4. Implementation and experimental results

4.1. Implementation environment

We have implemented our proposed algorithm in the Taiwan UniGrid platform [27]. There are more than 20 campuses participating in this project in Taiwan. The Taiwan UniGrid provides computing resources with the aid of a middleware called GT4 (Globus Toolkit of version 4) [29,35].

We select 25 computing nodes from the Taiwan UniGrid to implement the experiments, as shown in Fig. 3. The sites include Academia Sinica [36], National Center for High-Performance Computing (NCHC) [37], Hsing Kuo University (HKU) [38] and National Dong Hwa University (NDHU) [39].

The Portal, the Job Scheduler and the Information Server of the system architecture in Fig. 1 are in the NDHU campus.

Fig. 4 shows the CPU speed of each resource retrieved from NWS system. Machines from 1 to 4 are from NDHU, machines from 5 to 10 are from Academia Sinica, machines from 11 to 17 are from NCHC and machines from 18 to 25 are from HKU. After checking, the real CPU speed of each resource in Academia Sinica, NCHC and HKU is the same as the one retrieved from NWS.

In each resource, besides standard Linux scheduling algorithm [40], there is no local site scheduler such as portable batch scheduler [41] or load sharing facility [42]. If there were, CPU time measurement library call has to be embedded in the program to have the actual execution time.

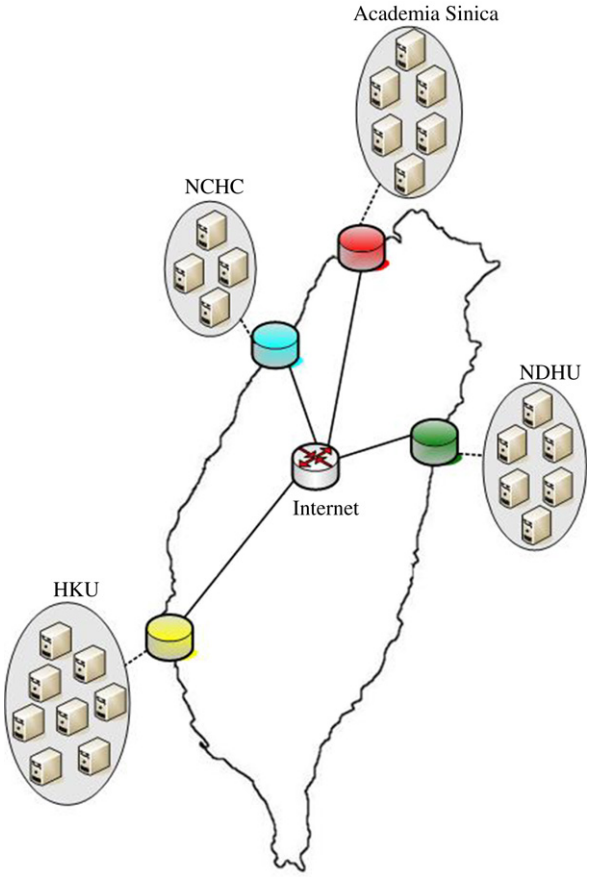

Fig. 3. Implementation environment.

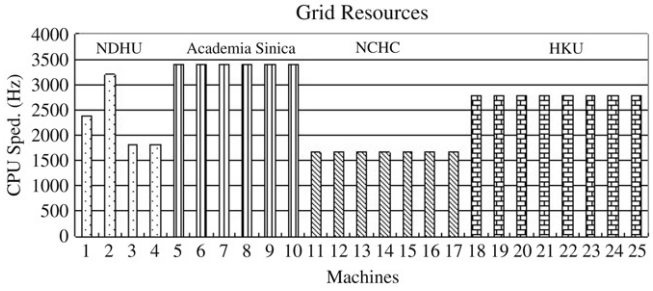

Fig. 4. CPU speed of each resource.

4.2. Implementation

This study focuses on the makespan and system load balance. Jobs in our experiments are to compute the product of matrices with different sizes.

When a client delivers a request, the system works as follows:

A. A client uses the client-interface to send a request that contains the total number of jobs, the size of matrix and the job scheduling algorithm to the Portal.
B. The Jobs Scheduler receives the message from the Portal and uses it as parameters for the BACO algorithm. The BACO algorithm starts to calculate the relevant parameters. At the same time, the Information Server would also provide the resource information to the Jobs Scheduler.
C. The BACO algorithm selects a resource for submitting the request (job) by finding the largest entry in the *PI* matrix among the available jobs to be executed. Then a local pheromone update is performed.
D. When a resource finishes a job, a global pheromone update is performed and the resource will send the final results back to the Portal. On receiving the execution results, the Portal would send it back to the client to be displayed on the client user interface.
E. Repeat Step A to Step D until all jobs are completed.

Table 3
Parameters of iACO

Parameter	Value
α	0.5
β	0.5
P	0.99
Ce (encouragement coefficient)	0.003
Cp (punishment coefficient)	0.002
C (coefficient of the load balancing factor)	0.4
Ke	750
Kp	250
K	500

The scheduling algorithms to be compared in the experiment include an improved ant algorithm in [13] (we call it iACO), the BACO, FPLTF [14], Dynamic FPLTF [15], Sufferage [15], and the random selection algorithm (random). The parameters of iACO algorithm are the same in [13] and the values of *Ke* (the coefficient of encourage factor), *Kp* (the coefficient of punishment factor) and *K* (the coefficient that is relevant to computation workload and communication quantity of the job) are shown in Table 3.

4.3. Experimental results

Two problems are simulated. The first is matrix multiplication. The second is linear programming using the GNU GLPK (GNU Linear Programming Kit) [43]. We simulate 1000 jobs which is the same as in [13]. For matrix multiplication, the size of the matrix is 500 × 500, 1000 × 1000, or 2000 × 2000. The size of each job depends on its matrix size. Take matrix size of 500 × 500 for example, we send 500 × 500 real numbers to the resource for execution and its job size is 500 × 500 × 4 bytes (976.5625 KB). For program execution time, since a straightforward matrix multiplication algorithm has a time complexity of $O(n^3)$, we use $2n^3$ (n additions plus n multiplications to obtain an entry) as the execution time estimate.

For linear programming, the size of the constraint is 200 × 200 (200 variables in 200 inequalities), 500 × 500, or 1000 × 1000. For a 200 × 200 problem, the program size is about (200 × 200 + 200) × 4 bytes. However, the programming execution time is hard to predict. From the time complexity of ellipsoid method [44], we use $200n^4$ as the execution time estimate for the GLPK linear programming program. The coefficient 200 stands for its complexity compared to a simple $O(n^4)$ algorithm.

4.3.1. Results of same matrix size multiplication and same size linear programming

We compare the number of job assignment on each resource, average execution time of jobs and the standard deviation of load of each method in the following experiments. The matrix sizes are all the same and include three cases: 500 × 500, 1000 × 1000, and 2000 × 2000. The linear programming sizes are also the same and contain three cases: 200 × 200, 500 × 500, 1000 × 1000.

Fig. 5 shows that the average execution time per job for matrix multiplication and Fig. 6 shows the linear programming case. BACO take less time to execute in each size of jobs than other methods because BACO let the resources which have good computing power and light load execute more jobs. That is the reason why BACO takes less time to execute jobs in each case in average.

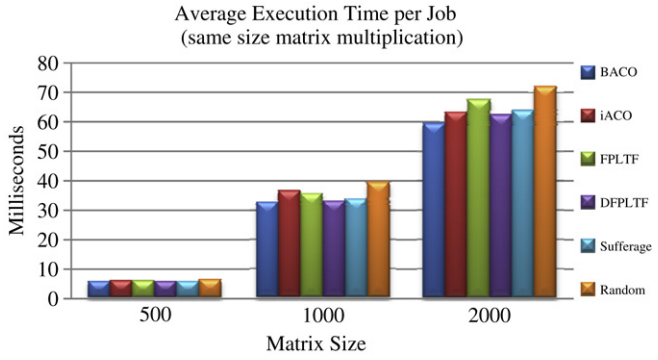

Fig. 5. Average execution time per job for matrix multiplication.

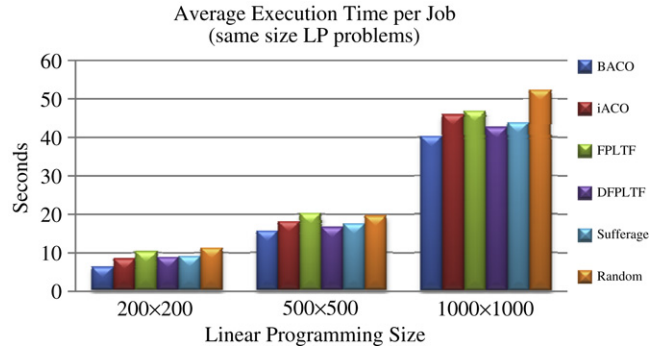

Fig. 6. Average execution time per job for linear programming.

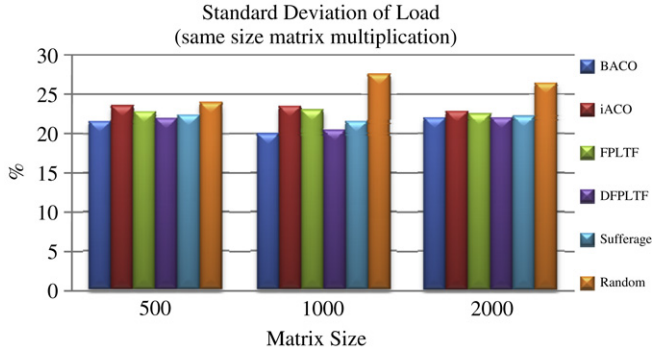

Fig. 7. Standard deviation of load for matrix multiplication.

Next we sample the load of each resource 10 times during the job execution and compare the standard deviation of load of each method independently. The standard deviation is given by:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}, \quad \text{for all } i$$

where σ is the standard deviation with the same unit as load (%), N is the number of resources, x_i is the load of resource i, and $\overline{x}$ is the average load of all resources.

If the standard deviation value of a method is small, it means that the difference of each load is small. The small standard deviation tells that the load of the entire system is balanced. The lower value the standard deviation has, the more load balanced the system is.

Figs. 7 and 8 show the standard deviation of load of each method. BACO has the smallest value. It means that the difference of resource load is small and the resources are load balanced.

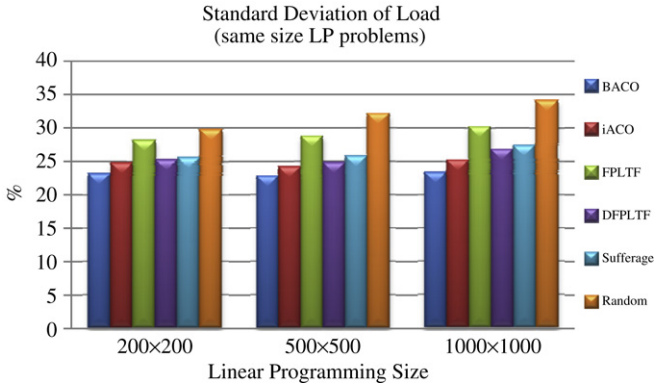

Fig. 8. Standard deviation of load for linear programming.

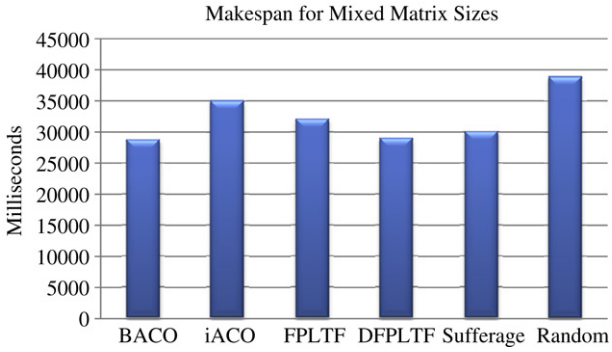

Fig. 9. Makespan of each method with mixed sizes for matrix multiplication.

4.3.2. Results of mixed sizes

The purpose of simulating the mixed size is that there may be many different jobs in the grid and we want to know how the BACO algorithm performs in such dynamic situation.

We choose 1000 jobs for execution and set the number of each size to one third of the total number of jobs. So the number of jobs is 333, 333, and 334, respectively for each size whether in matrix multiplication or in linear programming. We compare the makespan (total execution time) and the standard deviation of load in each method.

Figs. 9 and 10 show the makespan of each method with mixed sizes. We can find out that BACO uses less time to complete all jobs. In the case of mixed jobs, the pheromone update functions of BACO still work well. For iACO, it applies the encouragement and punishment methods. It changes each pheromone by variables defined by users and it ignores the real status of resources. If the resources with bad computing power never fail to execute jobs, they will always be encouraged and get more pheromone. Then iACO will assign more jobs to the bad resources which have higher pheromone and this will increase the total execution time of the given jobs. That is the reason why iACO has larger makespan than BACO.

Figs. 11 and 12 show the standard deviation of load in each method. In the case of mixed jobs, BACO still has the smallest standard deviation of load. Its resource selection depends on the resources status and the size of jobs; it means that BACO considers about the real status of resources. By the results, it does balance the load of the entire system.

By the experimental results, we can easily find out the BACO can achieve good system load balance in any situation and take less time to execute jobs. It means that BACO can handle different size jobs or same size jobs in grid and also keeps the system load more balanced and have better performances.

iACO may have bad performances because the prediction of status of resources in the pheromone update methods may be

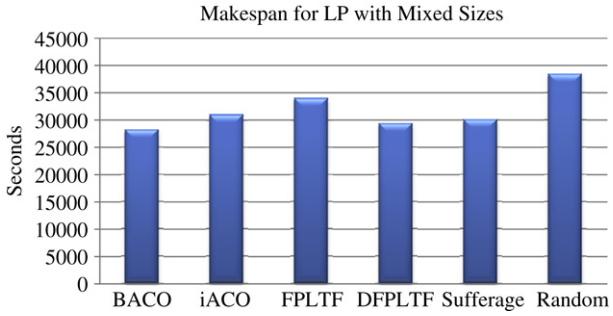

Fig. 10. Makespan of each method with mixed sizes for linear programming.

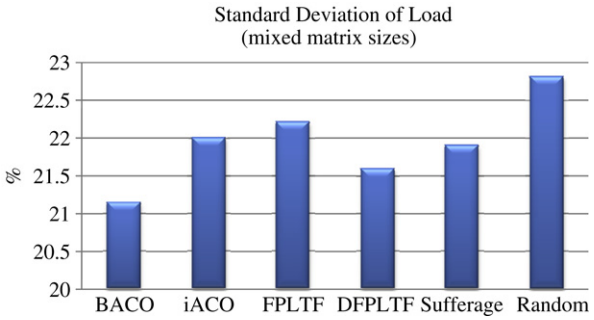

Fig. 11. Standard deviation of load of each method with mixed size for matrix multiplication.

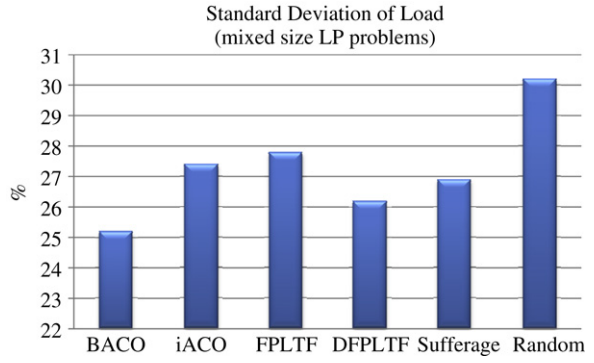

Fig. 12. Standard deviation of load of each method with mixed size for linear programming.

wrong. The prediction means that it uses user-defined variables to encourage or punish resources after the assignment of jobs or the completion of jobs. It may not work sometimes when the pheromone values do not match the real status of resources. So BACO can work better than iACO.

DFPLTF and Sufferage have similar performances and are also comparable to that of BACO. However, they do not consider the bandwidth issue and assume the data is readily available in the resources, which is sometimes not true.

FPLTF may have bad performance if the number of hard tasks is too many and it always assigns jobs to the fastest resources which may already have a heavy load. It will cause the system load to be unbalanced and take much time for job executions.

Random has the true random performance. It does not consider about the status of resources and the size of jobs and assign jobs to resources randomly.

5. Conclusions and future work

In this paper, we propose a BACO algorithm to choose suitable resources to execute jobs according to resources status and the size of given job in the Grid environment. The local and global pheromone update functions do balance the system load.

Local pheromone update function updates the status of the selected resource after jobs assignment. Global pheromone update function updates the status of each resource for all jobs after the completion of a job. It offers the Job Scheduler the newest information of all resource for the next jobs assignment. The experimental result shows that BACO is capable of balance the entire system load.

In future, we will study whether there are any other situations which we do not take into account on our definitions of the pheromone indicator or the pheromone update functions. We will also try to apply BACO algorithm to various grid computing applications. For example, instead of independent jobs, assume now we are scheduling workflows. That is, there are precedence relations among jobs. Then BACO has to be modified to include a synchronization scheme among resources. When a job is to be assigned to a resource for execution, we must be certain that all its precedent jobs running on other resources have been completed.

Finally, this paper focuses on the computing grid. We may redefine the pheromone indicator and pheromone update formulations for the data grid to consider the replica strategy to select or predict which resources have more storage or are suitable for file replications by their newest status in future.

Acknowledgements

This research is supported in part by ROC NSC under contract numbers 95-2422-H-259-001 and 94-2213-E-259-004.

References

[1] D.A. Reed, Grids, the TeraGrid, and Beyond, IEEE Computer 36 (1) (2003) 62–68.
[2] BOINC website, http://boinc.berkeley.edu/.
[3] D. Kondo, D.P. Anderson, J. McLeod, Performance evaluation of scheduling policies for volunteer computing, in: Proc. IEEE International Conference on e-Science and Grid Computing, Dec. 2007, pp. 415–422.
[4] Ruay-Shiung Chang, Jih-Sheng Chang, Shin-Yi Lin, Job scheduling and data replication on data grids, Future Generation Computer Systems 23 (7) (2007) 846–860.
[5] Yang Gao, Hongqiang Rong, Joshua Zhexue Huang, Adaptive grid job scheduling with genetic algorithms, Future Generation Computer Systems 21 (1) (2005) 151–161.
[6] EunJoung Byun, SungJin Choi, MaengSoon Baik, JoonMin Gil, ChanYeol Park, ChongSun Hwang, MJSA: Markov job scheduler based on availability in desktop grid computing environment, Future Generation Computer Systems 23 (4) (2007) 616–622.
[7] F. Dong, S.K. Akl, Scheduling algorithms for grid computing: State of the art and open problems, Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, Canada, January 2006.
[8] M. Dorigo, C. Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2–3) (2005) 243–278.
[9] M. Dorigo, Ant colony optimization, http://www.aco-metaheuristic.org.
[10] M. Dorigo, L.M. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 53–66.
[11] E. Salari, K. Eshghi, An ACO algorithm for graph coloring problem, in: Congress on Computational Intelligence Methods and Applications, 15–17 Dec. 2005, p. 5.
[12] Xiaoxia Zhang, Lixin Tang, CT-ACO—hybridizing ant colony optimization with cycle transfer search for the vehicle routing problem, in: Congress on Computational Intelligence Methods and Applications, 15–17 Dec. 2005, p. 6.
[13] H. Yan, X. Qin, X. Li, M.-H. Wu, An improved ant algorithm for job scheduling in grid computing, in: Proceedings of 2005 International Conference on Machine Learning and Cybernetics, vol. 5, 18–21 Aug. 2005, pp. 2957–2961.
[14] D. Saha, D. Menasce, S. Porto, Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures, Journal of Parallel and Distributed Computing 28 (1) (1995) 1–18.
[15] D. Paranhos, W. Cirne, F. Brasileiro, Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids, in: International Conference on Parallel and Distributed Computing (Euro-Par), in: Lecture Notes in Computer Science, vol. 2790, 2003, pp. 169–180.
[16] T. Stutzle, MAX-MIN Ant System for Quadratic Assignment Problems Technical Report AIDA-97-04, Intellectics Group, Department of Compute Science, Darmstadt University of Technology, Germany, July 1997.

[17] B. Bullnheimer, R.F. Hartl, C. Strauss, A new rank-based version of the ant system: A computational study, Central European Journal for Operations Research and Economics 7 (1) (1999) 25–38.
[18] E.D. Taillard, L.M. Gambardella, Adaptive memories for the quadratic assignment problem, Technical Report IDSIA-87-97, IDSIA, Lugano, Switzerland, 1997.
[19] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics, Part B 26 (1) (1996) 29–41.
[20] Kwang Mong Sim, Weng Hong, Sun, Multiple ant-colony optimization for network routing, in: Proceedings of the First International Symposium on Cyber Worlds, 6–8 Nov. 2002, pp. 277–281.
[21] J. Heinonen, F. Pettersson, Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem, Applied Mathematics and Computation 187 (2) (2007) 989–998.
[22] Jianfu Li, Wei Zhang, Solution to multi-objective optimization of flow shop problem based on ACO algorithm, in: Proceeding of 2006 International Conference Computational Intelligence and Security, vol. 1, Nov. 2006, pp. 417–420.
[23] E. Burke, G. Kendall, Silva Landa, R. O'Brien, E. Soubeiga, An ant algorithm hyperheuristic for the project presentation scheduling problem, IEEE Congress on Evolutionary Computing 3 (2005) 2263–2270.
[24] Walter, J. Gutjahr, M.S. Rauner, An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria, Computers & Operations Research 41 (3) (2007) 645–666.
[25] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, 7th edition, John Wiley & Sons, 2005.
[26] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing system, Journal of Parallel and Distributed Computing 59 (1999) 107–131.
[27] Taiwan unigrid project portal site, http://www.unigrid.org.tw.
[28] Network Weather Service (NWS), http://nws.cs.ucsb.edu/ewiki/.
[29] Globus Toolkit v4, http://www.globus.org/toolkit/downloads/4.0.4/.
[30] V. Sarkar, Determining average program execution times and their variance, in: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 1989, pp. 298–312.
[31] C.Y. Park, Predicting program execution times by analyzing static and dynamic program paths, Real-Time Systems 5 (1) (1993) 31–62.
[32] J. Engblom, A. Ermedahl, Modeling complex flows for worst-case execution time analysis, in: Proceedings of the 21st IEEE Real-Time Systems Symposium, 2000, pp. 163–174.
[33] F. Stappert, P. Altenbernd, Complete worst-case execution time analysis of straight-line hard real-time programs, Journal of Systems Architecture 46 (4) (2000) 339–355.
[34] Yuanyuan Zhang, Wei Sun, Yasushi Inoguchi, Predict task running time in grid environments based on CPU load predictions, Future Generation Computer Systems 24 (6) (2008) 489–497.
[35] The globus alliance, http://www.globus.org/.
[36] Academia sinica, http://www.sinica.edu.tw/.
[37] National Center for High Performance Computing, http://www.nchc.org.tw/.
[38] Hsing Kuo University of Management (HKU), http://www.hku.edu.tw.
[39] National Dong Hwa University (NDHU), http://www.ndhu.edu.tw.
[40] Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, O'reilly Media, Oct. 2000.
[41] R.L. Henderson, Job scheduling under the portable batch system, in: Lecture Notes in Computer Science, vol. 949, 1995, pp. 279–294.
[42] Load sharing facility, http://www.platform.com/Products/platform-lsf.
[43] GLPK, http://www.gnu.org/software/glpk/.
[44] Robert G. Bland, Donald Goldfarb, Michael J. Todd, Ellipsoid method, a survey, Operations Research 29 (6) (1981) 1039–1091.

Ruay-Shiung Chang received his B.S.E.E. degree from National Taiwan University in 1980 and his Ph.D. degree in Computer Science from National Tsing Hua University in 1988. He is now a professor in the Department of Computer Science and Information Engineering, National Dong Hwa University. His research interests include Internet, wireless networks, and grid computing. Dr. Chang is a member of ACM and IEICE, a senior member of IEEE, and founding member of ROC Institute of Information and Computing Machinery. Dr. Chang also served on the advisory council for the Public Interest Registry (www.pir.org) from 2004/5 to 2007/4.

Jih-Sheng Chang received his B.E. degree from the Department of Computer Science and Information Engineering, I-Shou University, Kaohsiung, Taiwan in 2002 and his M.S. degree from the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan in 2004. He is a Ph.D. candidate at the Department of Computer Science and Information Engineering at National Dong Hwa University currently. His academic research interests focus on wireless network technology and grid computing.

Po-Sheng Lin received his master degree from the department of computer science and information management, National Dong Hwa University, in 2007. He is currently working as an engineer in a telecommunication company in Taipei, Taiwan. His research interests include grid computing and wireless networks.