Learning Automata Based Algorithms for Mapping of a Class of Independent Tasks over Highly Heterogeneous Grids

S. Ghanbari and M.R. Meybodi

Soft Computing Laboratory,
Computer Engineering Department and Information Technology,
Amirkabir University, Tehran Iran
saeed_ghanbari@yahoo.com , meybodi@ce.aut.ac.ir

Abstract. Computational grid provides a platform for exploiting various computational resources over wide area networks. One of the concerns in implementing computational grid environment is how to effectively map tasks onto resources in order to gain high utilization in the highly heterogeneous environment of the grid. In this paper, three algorithms for task mapping based on learning automata are introduced. To show the effectiveness of the proposed algorithms, computer simulations have been conducted. The results of experiments show that the proposed algorithms outperform two best existing mapping algorithms when the heterogeneity of the environment is very high.

1 Introduction

Owing to advances in computational infrastructure and networking technology, construction of large-scale high performance distributed computing environment, known as *computational grid*, is now possible. Computational grid enables the sharing, selection, and aggregation of geographically distributed heterogeneous resources for solving large scale problems in science, engineering and commerce. Numerous efforts have been exerted focusing on various aspects of grid computing including resource specifications, information services, allocation, and security issues. A critical issue to meeting the computational requirements on the grid is the scheduling.

Ensuring a favorable efficiency over computational grid is not a straightforward task, where a number of issues make scheduling challenging even for highly parallel applications. Resources on the grid are typically shared and undedicated so that the contention made by various applications results in dynamically fluctuating delays, capricious quality of services, and unpredictable behavior, which further complicate the scheduling. Regarding to these hurdles, the scheduling of applications on computational grids have become a major concern of multitude efforts in recent years[9].

In mixed-machine *heterogeneous computing* (HC) environments like computational grids, based on application model characterization, platform model

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 681 – 690, 2005.
© Springer-Verlag Berlin Heidelberg 2005

characterization and mapping strategy characterization, there are various definitions for scheduling[6]. Ideal sorts of applications for computational grid are those composed of independent tasks, which tasks can be executed in any order and there is no inter-task communication (i.e. totally parallel) [1][12]. There are many applications of such feature including data mining, massive searches (such as key breaking), parameter sweeps, Monte Carlo simulations[2], fractals calculations (such as Mandelbrot), and image manipulation applications (such as tomographic reconstruction[3]). Computational grid platform model consists of different high-performance machines, interconnected with high-speed links. Each machine executes a single task at a time (i.e. no multitasking) in the order to which the tasks are assigned. The matching of tasks to machines and scheduling the execution order of these tasks is referred to as mapping. The general problem of optimally mapping tasks to machines in an HC suite has been shown to be NP-complete [11].

In this paper, we present three algorithms based on learning automata for mapping metatask over HC. Through computer simulations we show that the proposed algorithms outperform the best existing mapping algorithms when the heterogeneity of the environment is very high.

This paper is organized as follows: Section 2 discusses the related works. Section 3 introduces learning automata. Section 4 explains the model of the Grid and the definitions used in later sections. Section 5 introduces the proposed learning automata based algorithms. In Section 6, experimental results are discussed, and section 7 provides the conclusion.

2 Related Works

Existing mapping algorithms can be categorized into two classes[4]: on-line mode (immediate) and batch mode. In on-line mode, a task is mapped onto a host as soon as it arrives at the scheduler. In the batch mode, tasks are collected into a set that is examined for mapping at certain intervals called *mapping events*. The independent set of tasks that is considered for mapping at the mapping events is called a *metatask*. The on-line mode is suitable for low arrival rate, while batch-mode algorithms can yield higher performance when the arrival rate of tasks is high because there are a sufficient number of tasks to keep hosts busy between the mapping events, and scheduling is done according to the resource requirement information of all tasks in the set[4]. The objective of most mapping algorithms is to minimize *makespan*, where makespan is the time needed for completing the execution of a metatask. Minimizing *makespan* yields to higher throughput.

Reported batch mode heuristics are Min-Min, Max-Min, Genetic Algorithm (GA), Simulated Annealing (SA), Genetic Simulated Annealing (GSA), A^* search, Suffrage[5][4], and *Relative Cost* (RC) [7] . Experimental results show that among batch-mode heuristics, Min-Min and GA give lower *makespan* than other heuristics[5], and RC further outperforms both GA and Min-Min[7].

RC introduces two essential criteria for a high-quality mapping algorithm for heterogeneous computing systems: *matching* which is to better match the tasks and machines, and *load balancing* which is to better utilize the machines. It is shown that in order to minimize the makespan, matching and system utilization should be

maximized[7]. However, these design goals are in conflict with each other because mapping tasks to their first choice of machines may cause load imbalance. Therefore, the mapping problem is essentially a tradeoff between the two criteria. Two out of three proposed algorithms in this paper resolve mapping by optimizing matching and load balancing.

3 Learning Automata

Learning Automata are adaptive decision-making devices operating on unknown random environments. A Learning Automaton has a finite set of actions and each action has a certain probability (unknown to the automaton) of getting rewarded by the environment of the automaton. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction on the system. If the learning algorithm is chosen properly, then the iterative process of interacting on the environment can be made to result in selection of the optimal action. Learning Automata can be classified into two main families: fixed structure learning automata and variable structure learning automata (VSLA) [8]. In the following, the variable structure learning automata which will be used in this paper is described.

A VSLA is a quintuple $< \alpha, \beta, p, T(\alpha, \beta, p) >$, where α, β, and p are an action set with r actions, an environment response set, and the probability set p containing r probabilities, each being the probability of performing every action in the current internal automaton state, respectively. The function of T is the reinforcement algorithm, which modifies the action probability vector p with respect to the performed action and received response. If the response of the environment takes binary values learning automata model is P-model and if it takes finite output set with more than two elements that take values in the interval [0,1], such a model is referred to as Q-model, and when the output of the environment is a continuous variable in the interval [0,1], it is refer to as S-model. Assuming $\beta \epsilon [0,1]$, a general linear schema for updating action probabilities can be represented as follows. Let action i be performed then:

$$p_j(n+1) = p_j(n) + \beta(n)[b/(r-1) - bp_j(n)] - [1-\beta(n)]ap_j(n) \quad \forall j \quad j \neq i \tag{1}$$

$$p_i(n+1) = p_i(n) - \beta(n)bp_i(n) + [1-\beta(n)]a[1-p_i(n)] \tag{2}$$

where a and b are reward and penalty parameters. When $a=b$, the automaton is called L_{RP}. If $b=0$ the automaton is called L_{RI} and if $0<b<<a<1$, the automaton is called $L_{R\varepsilon P}$. For more Information about learning automata the reader may refer to [8].

4 Simulation Model

This section presents a general model of the computational grid. The environment consists of the heterogeneous suite of machines which will be used to execute the application. The scheduling system consists of the automata, and the model of the application and the HC suite of machines.

The application and HC suite of machines are modeled as the estimate of the expected execution time for each task on each machine, which is known prior to the execution and contained within a $\tau \times \mu$ *ETC* (Expected Time to Compute) *matrix,* where τ is the number of tasks and μ is the number of machines. One row of the *ETC* matrix contains the estimated execution times for a given task on each machine. Similarly, one column of the *ETC* matrix consists of the estimated execution times of a given machine for each task in the metatask. Thus, for an arbitrary task s_i and an arbitrary machine m_j, $ETC(s_i,m_j)$ is the estimated execution time of s_i on m_j.

We define $\psi^{(n)}(i)=j$ as a general mapping from the task domain $i=1,\ldots,\tau$ to the machine domain $j=1,\ldots,\mu$ at iteration n. The load of each machine, which is denoted by $\theta^{(n)}(j)$, is defined as the time taken to execute all the assigned tasks:

$$\theta^{(n)}(j) = \sum ETC(k,j), j = \psi^{(n)}(k) \; 1 \le k \le \tau \quad (3)$$

The maximum of $\theta^{(n)}(j)$, over $1\le j\le \mu$, is the metatask execution time, which is referred to as *makespan*, denoted by $T_\mu^{(n)}$.

5 Proposed Learning Automata Model

The learning automata model is constructed by associating every task s_i in the metatask with a variable structure learning automaton, which is represented by a 3-tuple$(a(i),\beta(i),A(i))$. Each action of an automaton is associated with a machine, and since the tasks can be assigned to any of the μ machines, the action set of all learning automata are identical. Therefore, for any task s_i, $1\le i\le \tau$, $a(i)=m_1,m_2,\ldots,m_\mu$($m_i$ is the ith machine), and $\beta(i)\epsilon[0,1]$, where $\beta(i)$ closer to 0 indicates that the action taken by the automaton of task s_i is favorable to the system, and closer to 1 indicates an unfavorable response. Reinforcement scheme used to update action probabilities of learning automata is L_{RI}.

To determine the goodness of an action taken by an automaton, we propose three different algorithms. The first algorithm calculates $\beta(i)$ for each automaton $A(i)$ according to the reduction made in *makespan* and the load of the selected machine. The second and third algorithms calculate the goodness of an action based on improvement made in matching and load balancing.

5.1 Algorithm No.1

The algorithm No.1 (A1) determines the $\beta^{(n)}(i)$ at iteration n for each automaton $A(i)$ by considering *makespan* and load of the chosen machine. Algorithm A1 interprets the environment as P-model; therefore $\beta^{(n)}(i)\epsilon\{0,1\}$. Makespan at iteration n may be greater, less than, or equal to makespan at iteration $n-1$. Similarly, load of the machine chosen by automaton $A(i)$ at iteration n may be greater, less than, or equal to load of the machine chosen by the automaton at iteration $n-1$. Therefore, regarding to makespan and the load of the chosen machine in two consecutive iterations, nine states are possible. To determine the $\beta^{(n)}(i)$, we associate a probability value to each nine possible state, which determines the probability of rewarding the chosen action. Probability one means that the chosen action will be rewarded. Table 1 shows the values, where *D*, *U* and *I* stand for decrease, remaining unchanged, and increase, respectively.

Table 1. Reward probability associated with each state

Makespan	*Load of chosen machine*	*Rewarding probability*
D	*D*	1
D	*U*	0.875
D	*I*	0.75
U	*D*	0.625
U	*U*	0.5
U	*I*	0.375
I	*D*	0.25
I	*U*	0.125
I	*I*	0

Algorithm Al is suitable for situations that the information used to evaluate the environment response is the load of machines.

5.2 Algorithm No.2

As mentioned in section 2, it is shown that to minimize the *makespan*, matching and system utilization must be maximized. Algorithm No.2 (A2) evaluates the response to the learning automata by considering these two criteria. Matching of tasks and machines can be measured by a parameter, *matching proximity*, which is defined as follows:

$$\eta = \frac{\sum_{1 \le i \le \tau} ETC(i, \psi_{\min}(i))}{\sum_{1 \le i \le \tau} ETC(i, \psi(i))} \tag{4}$$

where $\eta \le 1$, and $\psi_{min}(i)$ is the ideal matching. Ideal matching is defined as executing every task on the machine with the shortest execution time. It is defined as follows:

$$\psi_{\min}(i) = j \text{ such that } ETC(i, j) = \min_{1 \le q \le \mu} ETC(i, q) \tag{5}$$

when $\eta=1$, we have the ideal matching. *System utilization* is defined as follows:

$$\delta = \frac{\sum_{1 \le j \le \mu} \theta(j)}{\mu \times T_{\mu}} \tag{6}$$

When the system is completely balanced, $\delta=1$; otherwise $\delta<1$.

Algorithm A2 reduces the mapping problem to an optimization problem with matching proximity and system utilization as objective functions. Algorithm A2 interprets the environment as S-model; therefore, $\beta^{(n)}(i)$ is in $[0,1]$.

To evaluate the contribution of each automaton to the improvement of matching and system utilization, we define two parameters, *partial contribution to matching*(PCM), and *partial contribution to load balancing*(PCL). Input to each automaton is a linear combination of PCM (denoted by $\eta^{(n)}(i)$), and PCL (denoted by $\delta^{(n)}(i)$):

$$\beta^{(n)}(i) = \eta^{(n)}(i)\lambda_{\eta} + \delta^{(n)}(i)\lambda_{\delta} \text{ where } \lambda_{\eta} + \lambda_{\delta} = 1 \tag{7}$$

λ_η and λ_δ are weights associated with PCM and PCL, respectively. PCM for each automaton *A(i)* at iteration *n* is evaluated as:

$$\eta^{(n)}(i) = \frac{ETC(i,\psi^{(n)}(i)) - ETC(i,\psi_{\min}(i))}{ETC(i,\psi_{\max}(i)) - ETC(i,\psi_{\min}(i))} \tag{8}$$

where $\psi_{max}(i)$ is the *worst matching* which is defined as mapping each task to a machine with the longest execution time; it is defined below

$$\psi_{\max}(i)=j \text{ such that } ETC(i,j) = \max_{1\le q\le \mu} ETC(i,q) \tag{9}$$

The closer $\eta^{(n)}(i)$ to 0, the more favorable the response from the environment as far as the matching is concerned. In the case that the automaton selects the machine with the worst matching, $\eta^{(n)}(i)$ is evaluated to 1.

PCL for each automaton *A(i)* at iteration *n* is evaluated as:

$$\partial^{(n)}(i) = \frac{\theta^{(n)}(\psi^{(n)}(i))}{T_\mu^{(n)}}\left(1 - e^{-\frac{1}{2}\left(\frac{\delta^{(n)}-1}{0.1}\right)^2}\right) \tag{10}$$

The former part of the above expression is close to 0 when the chosen machine has a load less than the maximum load. In this way, the learning automata are encouraged to choose machines with low loads, thus, they are guided in a way to decrease the distance between the maximum load and the minimum load. The latter part of the expression is a Gaussian function, which gets closer to 0 as the system utilization increases; therefore, when the load is relatively balanced, PCL of each automaton is close to 0. Unlike algorithm A1, algorithm A2 requires information about the estimation of execution time of each task on each machine.

5.3 Algorithm No.3

Algorithm No.3 (A3) interprets the environment as a Q-Model environment. Like algorithm A2, it uses matching proximity and system utilization as objective functions. PCL and PCM are evaluated in the same way as algorithm A2, and used to produce the environment response. But, in algorithm A3, PCL and PCM are interpreted as probabilities, where PCL determines the probability that the learning automaton receives unfavorable response as far as system utilization is concerned, and PCM determines the probability that the learning automaton receives unfavorable response as far as matching is concerned. The environment response is evaluated as below:

$$\beta^{(n)}(i) = I(\eta^{(n)}(i))\lambda_\eta + I(\delta^{(n)}(i))\lambda_\delta \text{ where } \lambda_\eta + \lambda_\delta = 1 \tag{11}$$

λ_η and λ_δ are the weights associated with PCM and PCL, respectively. *I(p)* is an indicator function which returns 1 with the probability of *p*, and 0 with the probability of *1-p*. Therefore, the input to each automaton $\beta(i)$ is in $\{0,\lambda_\eta,\lambda_\delta,1\}$. In contrast to algorithm A2, algorithm A3 evaluates environment response stochastically, which allows the learning automata to jump local minimums in their search space.

6 Experiments

In this section the proposed algorithms are tested and compared with algorithms Min-Min and RC because these two algorithms are the best existing algorithms. For the simulation studies, *ETC* matrices were generated using the method presented in [4]. Initially, a $\tau \times 1$ baseline column vector, *B*, of floating point values is created. Let ω_b be the upper bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, $x_b^i \epsilon [1, \omega_b)$, and letting $B(i)=x_b^i$ for $1 \leq i \leq \tau$. Next, the rows of the *ETC* matrix are constructed. Each element $ETC(s_i,m_j)$ in row *i* of the *ETC* matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $x_r^{i,j}$, which has an upper bound of ω_r. This new random number, $x_r^{i,j} \epsilon [1, \omega_r)$, is called a row multiplier. One row requires μ different row multipliers, $1 \leq j \leq \mu$. Each row *i* of the ETC matrix can then be described as $ETC(s_i,m_j) = B(i) \times x_r^{i,j}$, for $1 \leq j \leq \mu$. (The baseline column itself does not appear in the final ETC matrix.) This process is repeated for each row until the $\tau \times \mu$ *ETC* matrix is full. Therefore, any given value in the ETC matrix is within the range $[1, \omega_b \times \omega_r)$.

The amount of variance among the execution times of tasks in the metatask for a given machine is defined as task heterogeneity. Task heterogeneity is varied by changing the upper bound of the random numbers within the baseline column vector. High task heterogeneity was represented by ω_b=3000 and low task heterogeneity used ω_b=100. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Machine heterogeneity was varied by changing the upper bound of the random numbers used to multiply the baseline values. High machine heterogeneity values were generated using ω_r=1000, while low machine heterogeneity values used ω_r=10. The ranges were chosen to reflect the fact that in real situations there is more variability across execution times for different tasks on a given machine than the execution time for a single task across different machines.

Different *ETC* matrix consistencies were used to capture more aspects of realistic mapping situations. An *ETC* matrix is said to be *inconsistent* if the *ETC* matrices are kept in the unordered, random state in which they were created. The *ETC* matrix indicates *consistent* characteristics if a machine *j* executes any task *i* faster than machine *k*, then machine *j* executes all tasks faster than machine *k*. The consistent matrix can be obtained by sorting every row of the matrix independently. Between two special situations, a *semi-consistent* matrix represents a partial ordering among the machine/task execution times. For the *semi-consistent* matrix used here, the row elements in even columns of row *i* are extracted, sorted and replaced in order, while the row elements in odd columns remain unordered.

Twelve combinations of ETC matrix characteristics are possible: high or low task heterogeneity, high or low machine heterogeneity, and one type of consistencies (consistent, inconsistent, or semi-consistent). Among the twelve combinations the most heterogeneous environment is modeled with inconsistent, high task and machine heterogeneous *ETC*, and correspondingly the least heterogeneous environment is modeled with consistent, low task and machine heterogeneous *ETC*. Other combinations are between these two extremes. In charts presented in this section, Low and High task/machine heterogeneity are abbreviated to LoLo and HiHi, respectively.

All results reported here are averaged over 50 trials, and done for 200 tasks and 20 machines. The makespan for each experiment is normalized with respect to the benchmark heuristic, which is RC. Unless stated, the learning automata model used in the experiments is L_{RI} with *a=0.01* for algorithm A1 and *a=0.001* for algorithms A2 and A3. For algorithms A2 and A3, the weights λ_η and λ_δ are set to 0.4 and 0.6, respectively, for inconsistent environment, and set to 0.05 and 0.95 for semi-consistent and consistent environments. Matching weightage is set to a smaller value than system utilization weightage in semi-consistent and consistent environments, because in consistent environments all tasks have the same first choice for matching, the fastest machine. There is the same situation in a semi-consistent environment because of its consistent sub-matrix. Therefore, the decisive factor in gaining a better makespan is to maximize system utilization rather than matching proximity. Termination condition is met when, no change in makespan is made for 1500 consecutive iterations, or number of iterations exceeds 500000.

In Figure 1, three proposed algorithms are compared with Min-Min and RC in term of normalized makespan for different heterogeneity and consistency. For inconsistent environment, it can be noted that all three proposed algorithms outperform both RC and Min-Min. For high machine/task heterogeneity, makespan resulted by algorithm A3 is 21 percent less than the makespan resulted from RC. Algorithm A2 performs slightly better than algorithm A1, and algorithm A3 performs better than algorithms A1 and A2. For semi-consistent environment, all three proposed algorithms outperform Min-Min. Algorithms A1 and A3 perform better than RC for high task/machine heterogeneity; however, algorithm A2 fails to outperform RC. Except algorithm A3, the other two algorithms perform worse than RC for low task/machine heterogeneity. For consistent environment, it can be stated that RC and Min-Min performs better than the algorithms proposed in this paper.

Results shown in Figure 2 indicate the fact that the proposed algorithms perform signifycantly better than both RC and Min-Min for inconsistent environments, while they fails to perform better than RC and Min-Min for consistent environment. For semi-consistent environment whose heterogeneity is between consistent and inconsistent, learning automata outperforms Min-Min, but performs very closely to RC. Therefore, proposed algorithms operate better in environments with higher level of heterogeneity.

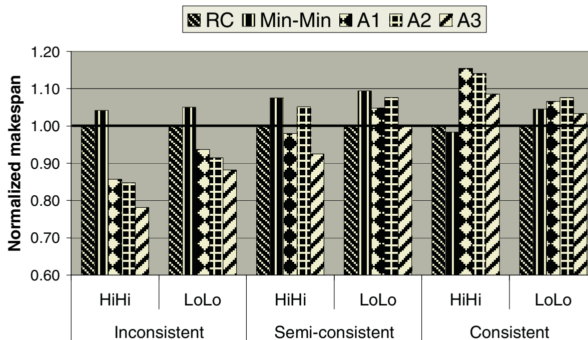

Fig. 1. Comparison of the proposed algorithm with RC and Min-Min for different consistency and heterogeneity

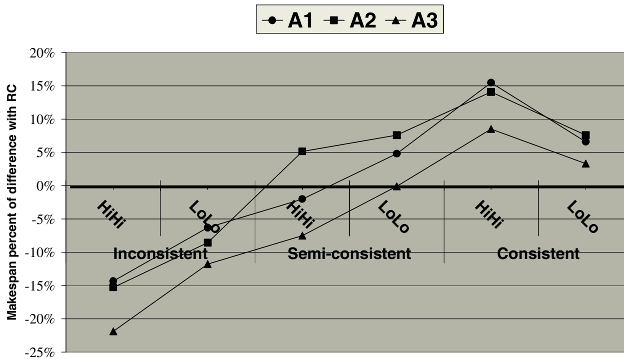

Fig. 2. Difference of makespan with RC for different consistency and heterogeneity

As expected, algorithm A3 performs better than algorithm A2 because it can avoid trapping in local minimums. Observing the results of the experiments, it is evident that algorithm A1 performs very close to and even better than algorithm A2 although it has a completely different reward criterion. It is worth mentioning that in contrast to algorithms A2 and A3 which use detailed information of expected run time of each task on each machine to guide learning automata, algorithm A1 ignores such information and guide learning automata blindly.

The other important issue to consider is the computational cost of finding a mapping using each proposed algorithm. On average, algorithm A2 finds a mapping in about 39000 iterations, while algorithms A1 and A3 needs 12 times more. Setting reward parameter (a) to 0.01 for algorithm A2 but 0.001 for algorithms A1 and A3 may account for faster convergence of A2. However, each algorithm is compared with others by setting learning parameter to a value that yields best result.

7 Conclusion

In this paper, we presented three algorithms based on learning automata for mapping a set of independent tasks over computational grid. The studied computational grid was modeled as a heterogeneous computing environment, and the objective of the proposed algorithm was to assign independent tasks to machines in a way to minimize *makespan*. Through experiments, we showed that for high heterogeneous environments, i.e. inconsistent environments, the proposed algorithms outperform two best existing mapping algorithms.

References

[1] A. L. Rosenberg, Optimal scheduling for cycle-stealing in a network of workstations with a bag-of-tasks workload, *IEEE Trans. Parallel Distributed Systems*, 13(2), 2002, 179-191.

[2] H. Casanova, T.M. Bartol, J. Stiles, and F. Berman, Distributing MCell simulations on the grid, *Int'l J. High Performance Computing Applications*, 15 (3), 2001, 243–257.
[3] S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M. Su, C. Kesselman, S. Young, and M. Ellisman, Combining workstations and supercomputers to support grid applications: the parallel tomography experience, *IEEE Proc. 9th Heterogeneous Computing Workshop*, 2000, 241–252.
[4] M. Macheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, *J. Parallel Distributed Computing*, 59 (2), 1999, 107–131.
[5] T. D. Braun, H. J. Siegel, and N. Beck, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *J. Parallel and Distributed Computing*, 61, 2001, 810-837.
[6] T. D. Braun, H. J. Siegel, et al., Taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems, *Proc. 17th IEEE Symposium on Reliable Distributed Systems*, 1998, 330-335.
[7] Min-You Wu and Wei Shu, A high-performance mapping algorithm for heterogeneous computing systems*, Proc. 15th Int'l Parallel and Distributed Processing Symposium (IPDPS'01)*, 2001.
[8] K. Narendra and M. A. L. Thathachar, "Learning Automata: An Introduction," Prentice Hall, Englewood Cliffs, New Jersey, 1989.
[9] F. Berman, High-performance schedulers, in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C Kesselman, eds., Morgan Kaufmann, San Francisco, CA, 1999, 279-310.
[10] H. Chen and M. Maheswaran, Distributed dynamic scheduling of composite tasks on grid computing systems, *Proc. Int'l Parallel and Distributed Processing Symposium (IPDPS'02),* 2002.
[11] O. H. Ibarra and C. E. Kim, Heuristic Algorithms for scheduling independent tasks on non-identical processors, *J. ACM,* 24(2), 1977, 280-289.
[12] C. Weng and X. Lu, Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid, *J. Future Generation Computer Systems*, Elsevier, 2003.