

# A Mapping Algorithm for Highly Heterogeneous Computational Grid: A Learning Automata Approach

S. Ghanbari M. R. Meybodi

Computer Engineering Department and Information Technology

Amirkabir University

Tehran Iran

saeed\_ghanbari@yahoo.com, meybodi@ce.aut.ac.ir

## Abstract

*The computational grid provides a platform for exploiting various computational resources over wide area network. One concern 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, a mapping algorithm based on learning automata is proposed. The results of experiments show that the proposed algorithm can obtain better performance compared to the two best existing algorithms in high task and machine heterogeneous environments.*

**Keywords:** Computational grid; Metatask; Mapping; Learning automata; Heterogeneous computing

## 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 characterization and mapping strategy characterization, there are various definitions for scheduling [6]. Ideal sorts of applications for computational grid are those composed of independent subtasks (called metatask), which subtasks can be executed in any order and there is no inter-task communication (i.e. totally parallel)[1]. 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 size of the metatask and the number of

machines in the HC suite are static and known beforehand. 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 a novel algorithm based on learning automata for mapping metatask over HC system. Through several experiments we show that the proposed algorithm outperforms 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 the learning automata. Section 4 explains the model of the Grid and the definitions used in later sections. Section 5 introduces the proposed learning automata algorithm and the heuristics proposed for guiding each automaton. In Section 6, experimental results are discussed and in section 7 we summarize the work and examine 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 of tasks 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 will be 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.

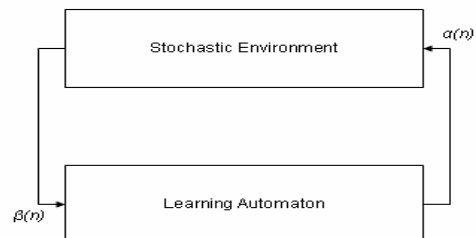
Minimum Completion Time (MCT), Minimum Execution Time (MET), Dual, and *k*-Percent Best (KPB) are among well known existing on-line mode heuristics [5]. Reported batch mode heuristics are Min-Min, Max-Min, Genetic Algorithm (GA), Simulated Annealing (SA), Genetic Simulated Annealing (GSA), A\* search, Suffrage [4,5], and Relative Cost (RC)[7]. Experiment results from [5] show that among batch-mode heuristics, Min-Min and GA give lower *makespan*. [7] proposes *Relative Cost* (RC) heuristic which further outperforms both GA and Min-Min.

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. The objective of mapping is to minimize the makespan that in a homogeneous system, it is equivalent to maximizing load balancing. In a heterogeneous system, in addition to load balancing, a task should be mapped to a machine that can execute the task fastest. An ideal algorithm should satisfy both criteria simultaneously. 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. A good algorithm must balance between matching proximity and system utilization. The algorithm proposed in this paper exploits learning automata to find mappings by optimizing matching and load balancing criteria simultaneously.

## 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 for 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. Figure 1 illustrates how a stochastic automaton works in feedback connection with a random environment. 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.



**Figure 1. The interaction between learning automata and environment**

A VSLA is a quintuple  $\langle a, \beta, p, T(a, \beta, p) \rangle$ , where  $a, \beta, p$  are an action set with  $s$  actions, an environment response set and the probability set  $p$  containing  $s$  probabilities, each being the probability of performing every action in the current internal automaton state, respectively. 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 referred to as S-model. The function of  $T$  is the reinforcement algorithm, which modifies the action probability vector  $p$  with respect to the performed action and received response. Assume  $\beta \in [0,1]$ . A general linear schema for updating action probabilities can be represented as follows. Let action  $i$  be performed then

$$\begin{aligned} p_j(n+1) &= p_j(n) + \beta(n)[b/(r-1) \\ &- bp_j(n)] - [1 - \beta(n)]ap_j(n) \quad \forall j \quad j \neq i \\ p_i(n+1) &= p_i(n) - \beta(n)bp_i(n) \\ &+ [1 - \beta(n)]a[1 - p_i(n)] \end{aligned} \quad (1)$$

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  $L_{ReP}$ . 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. Figure 2 shows the schematic representation of the environment. The environment consists of the heterogeneous suite of machines which will be used to execute the application. The scheduling system consists of the automata model, 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  $\tau$  is the number of tasks and  $\mu$  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  $m_j$ . The  $ETC(s_i, m_j)$  entry could be assumed to include the time to move

the executables and data associated with task  $s_i$  from their known source to machine  $m_j$ . For cases when it is impossible to execute task  $s_i$  on machine  $m_j$  (e.g., if specialized hardware is needed), the value of  $ETC(s_i, m_j)$  is set to infinity.

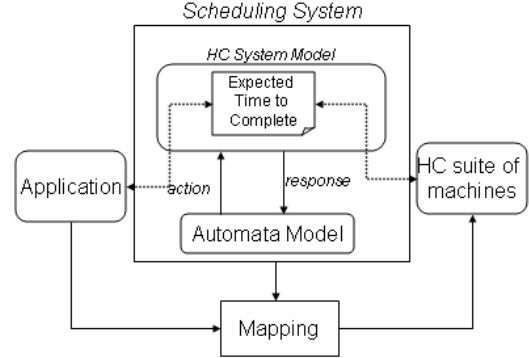


Figure 2-Model of the Grid

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

$$\theta(j) = \sum ETC(k, j), j = \psi(k) \quad 1 \leq k \leq \tau \quad (2)$$

The maximum  $\theta(j)$  value, over  $1 \leq j \leq \mu$ , is the metatask execution time, which is referred to as the *makespan*. It is shown that to minimize the *makespan*, matching and system utilization must be maximized [7]. Matching of tasks and machines can be measured by a parameter, *matching proximity*. When every task is executed on the machine with the shortest execution time, it is defined as the ideal matching which is represented as  $\psi_{\min}(i)$ ,  $1 \leq i \leq \tau$ , and evaluated as shown in equation 3.

$$\begin{aligned} \psi_{\min}(i) &= j \text{ such that} \\ ETC(i, j) &= \min_{1 \leq q \leq \mu} ETC(i, q) \end{aligned} \quad (3)$$

Correspondingly, when each task is mapped to a machine with the longest execution time, it is defined as the worst matching which is represented as  $\psi_{\max}(i)$ ,  $1 \leq i \leq \tau$  and evaluated as shown in equation 4.

$$\begin{aligned} \psi_{\max}(i) &= j \text{ such that} \\ ETC(i, j) &= \max_{1 \leq q \leq \mu} ETC(i, q) \end{aligned} \quad (4)$$

Thus, *matching proximity* is defined as follows:

$$\eta = \frac{\sum_{1 \leq i \leq \tau} \psi_{\min}(i)}{\sum_{1 \leq i \leq \tau} \psi(i)} \quad (5)$$

where  $0 < \eta \leq 1$ . When  $\eta = 1$ , we have the ideal matching.

Load balance can be measured by *system utilization*:

$$\delta = \frac{\sum_{1 \leq i < \mu} ETC(i, \Psi(i))}{\mu \times T_{\mu}} \quad (6)$$

which  $T_{\mu}$  is the makespan. When the system is completely balanced,  $\delta = 1$ ; otherwise  $\delta < 1$ . Thus, the makespan of a mapping can be expressed as:

$$T_{\mu} = \frac{\sum_{1 \leq i \leq \tau} \psi_{\min}(i) / \mu}{\eta \times \delta} \quad (7)$$

where the  $\sum_{1 \leq i \leq \tau} \psi_{\min}(i) / \mu$  is the lower-bound of the makespan. Therefore, to minimize the makespan,  $\eta$  and  $\delta$  must be maximized.

## 5. Proposed Learning Automata Based Mapping Algorithms

We propose a new mapping algorithm by means of learning automata. The mapping problem is reduced to an optimization problem with matching proximity and system utilization as objective functions.

The learning automata model is constructed by associating every task  $s_i$  in the metatask with a variable structure learning automaton, denoted by a 3-tuple  $(a^{s_i}, \beta^{s_i}, A^{s_i})$ . Since the tasks can be assigned to any of the  $\mu$  machines ( $\mu$  is the number of machines), the action set of all learning automata are identical. It is assumed that the environment is S-model, so the response to the learning automata is in interval  $[0, 1]$ . Therefore, for any task  $s_i$ ,  $1 \leq j \leq \tau$  ( $\tau$  number of tasks),  $a^{s_i} = m_1, m_2, \dots, m_{\mu}$  ( $m_i$  is the  $i^{\text{th}}$  machine) and  $\beta^{s_i} \in \{0, 1\}$ , where  $\beta^{s_i}$  closer to 0 indicates that the action taken by the automaton associated with task  $s_i$  is favorable to the system, and closer to 1 indicates an unfavorable response.

Input to each automaton is a linear combination of two parameters, *partial contribution to matching*

(PCM denoted by  $\eta(i)$ ), and *partial contribution to load balancing* (PCL denoted by  $\delta(i)$ ):

$$\beta^{s_i} = \eta(i)\lambda_{\eta} + \delta(i)\lambda_{\delta}, \quad \lambda_{\eta} + \lambda_{\delta} = 1 \quad (8)$$

$\lambda_{\eta}$  and  $\lambda_{\delta}$  are the weights associated with PCM and PCL, respectively.

PCM for each automaton  $A^{s_i}$  is calculated as:

$$\eta(i) = \frac{ETC(i, \psi(i)) - \psi_{\min}(i)}{\psi_{\max}(i) - \psi_{\min}(i)} \quad (9)$$

$\eta(i)$  closer to 0 represents a favorable response according to the matching. On the contrary, in the case that the automaton selects the machine with the worst matching  $\eta(i)$  equals to 1.

We have proposed and tested two heuristics for evaluating PCL. The first heuristic compares the load of each machine with the average load of all machines in HC suite. Higher reward is granted to those learning automata that choose the machines with load near the average. In this way, learning automata are guided to choose machines with average load:

$$\delta(i) = \frac{|\theta(\psi(i)) - \theta_{AVG}|}{\text{Max}_{1 \leq j \leq \mu} |\theta(j) - \theta_{AVG}|} \quad (10)$$

where

$$\theta_{AVG} = \frac{\sum_{j=1}^{\mu} \theta(j)}{\mu} \quad (11)$$

When the load is relatively balanced, the maximum distance between the load of machines and the average load is a small value. Thus, as the load gets more balanced;  $\delta(i)$  tends to higher values. Thus, even if an automaton chooses a machine with a load close to the average load receives penalty. This makes the learning automata convergence more difficult. As another deflection, the heuristic is in favor of high load machines. By numerous experiments, we learned that in certain situations, no task is assigned to a machine while others are heavily overloaded. In fact, when the average load is close to load of machines with heavy loads, and thus the machines with loads close to zero have greater distance from average load, PCL of automata choosing these machines are closer to 1 than PCL of automata choosing machines with higher loads. In consequence, machines with lower loads have less chance to be selected.

In order to address mentioned defections, another heuristic for evaluating PCL is proposed, shown in equation 12.

$$\partial(i) = \frac{\theta(\psi(i))}{T_\mu} (1 - e^{-\frac{1}{2}(\frac{\delta-1}{0.1})^2}) \quad (12)$$

where  $T_\mu$  is the *makespan*. The former part of the above expression is close to 0 when the chosen machine has a load less than the maximum load. Thus, the heuristic encourages the learning automata to choose machines with low loads. This guides the learning automata to decrease the distance of maximum load and minimum load. The latter part of the expression is a Gaussian function. It closes to 0 as the system utilization increases; therefore, when the load is relatively balanced, PCL of each automaton tends to 0. Throughout experiments, we used the second proposed heuristic for PCL.

## 6. Experiments

In this section the proposed algorithm is tested and compared with Min-Min and RC because these two algorithms are the best existing algorithms[7]. For the simulation studies, characteristics of the *ETC* matrices were varied in an attempt to represent a range of possible HC environments. The *ETC* matrices used were generated using the following method[4]. Initially, a  $\tau \times 1$  baseline column vector,  $B$ , of floating point values is created. Let  $\omega_b$  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,  $x_b^i \in [1, \omega_b)$ , 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  $\omega_r$ . This new random number,  $x_r^{i,j} \in [1, \omega_r)$ , is called a row multiplier. One row requires  $\mu$  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  $[1, \omega_b \times \omega_r)$ .

To generate different mapping scenarios, the characteristics of the *ETC* matrix were varied based

on several different methods. 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 is represented by  $\omega_b = 3000$  and low task heterogeneity uses  $\omega_b = 100$ . 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 is generated using  $\omega_r = 1000$ , while low machine heterogeneity values uses  $\omega_r = 10$ . The ranges are 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 are 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 are 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, where inconsistent *ETC* represents more heterogeneity than semi-consistent and consistent *ETC* represents less heterogeneity than semi-consistent. The results reported here are averaged over 50 different runs. All experiment results are for 200 tasks and 20 machines. Unless stated, the learning automata model used in the experiments is  $L_{RI}$  with  $a=0.01$ . The weights  $\lambda_\eta$  and  $\lambda_\delta$  are set to 0.4 and 0.6 respectively for inconsistent environment, and

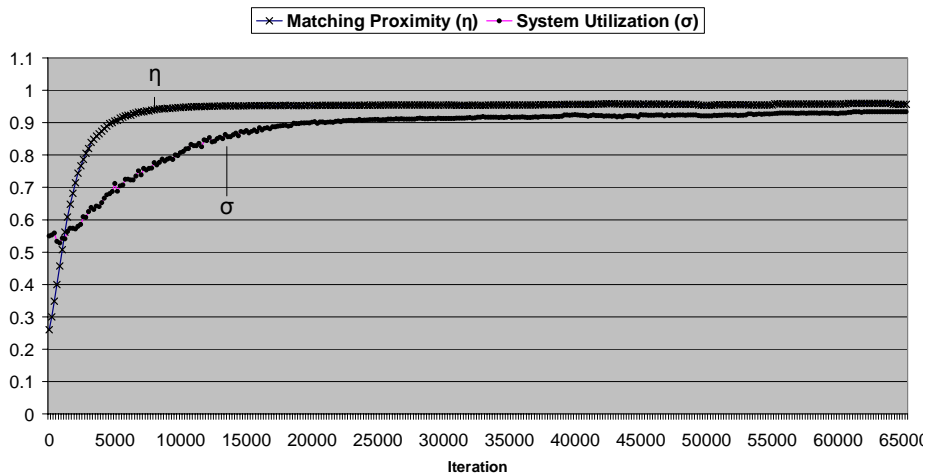
set to 0.05 and 0.95 for semi-consistent and consistent environments. Matching weightage is set to a very smaller value than system utilization weightage in semi-consistent and consistent environments, because in consistent environments all tasks have the same first choice for matching, the fastest machine. There is the same problem in a semi-consistent environment with its consistent sub-matrix. Therefore, the decisive factor in gaining a better makespan is to maximize system utilization rather than matching proximity. Unless stated, the comparison between the heuristics is done over makespan.

Experimental results for inconsistent environment are tabulated in Table 1. The results show that the proposed algorithm gives lower makespan, higher system utilization and matching proximity than both RC and Min-Min. The proposed learning automata algorithm improves RC by about 7 percent and Min-Min by about 13 percent for low task/machine heterogeneity, while outperforms RC by about 15% and makespan of Min-Min by about 18 percent for high task/machine heterogeneity. The results also show that as the heterogeneity increases, the proposed algorithm performs better in contrast to two other algorithms. In Figure 3, the trend of maximizing the system utilization and matching proximity is shown.

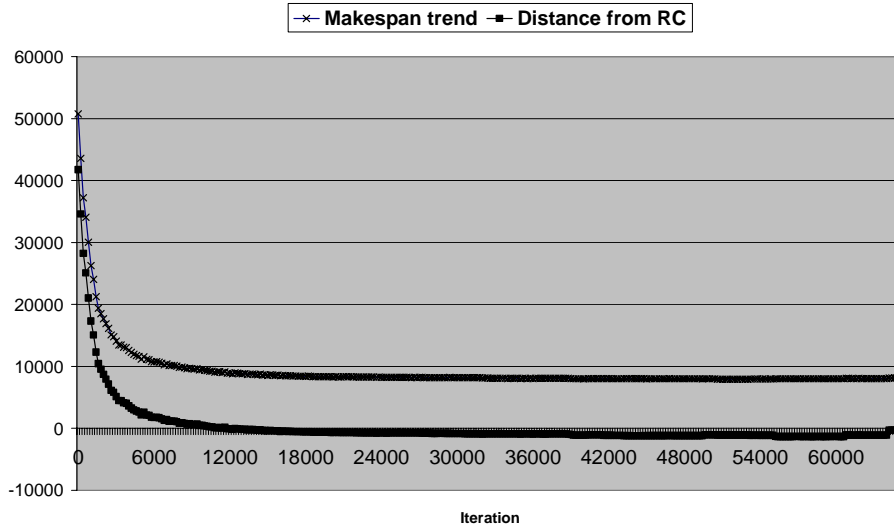
Figure 4 depicts the trend of reducing makespan with a comparison to RC and Min-Min for low task and machine heterogeneity.

**Table 1. Comparison: Inconsistent environment**

Heterogeneity	Heuristic	Makespan	Matching proximity	System utilization	Iteration
LoLo	Min-Min	9432	95.5%	79.2%	-
	RC	8980	96.0%	83.0%	-
	LA	8210	95.4%	91.0%	35363
HiLo	Min-Min	28280	95.5%	79.2%	-
	RC	26924	96.0%	83.0%	-
	LA	24708	95.5%	90.6%	34842
LoHi	Min-Min	410363	94.0%	63.1%	-
	RC	393660	92.1%	67.2%	-
	LA	336216	86.9%	83.3%	31120
HiHi	Min-Min	1229373	94.1%	63.1%	-
	RC	1180466	92.2%	67.1%	-
	LA	1000139	86.9%	83.6%	32369



**Figure 3. Matching proximity & system utilization improvement for low task/machine heterogeneity**



**Figure 4. Trend of minimizing makespan for low task/machine heterogeneity**

Experimental results for semi-consistent environment are tabulated in Table 2 and show that the proposed algorithm outperforms Min-Min. The proposed algorithm results are very close to RC. LA is about 2 percent better than Min-Min and about 7 percent worse than RC.

**Table 1. Comparison: Semi-consistent environment**

		Makespan	Matching proximity	System utilization	Iteration
LoLo	Min-Min	12007	74.3%	79.9%	-
	RC	10977	70.0%	92.8%	-
	LA	11808	65.6%	92.0%	37142
HiLo	Min-Min	36332	74.4%	79.2%	-
	RC	33070	70.0%	92.4%	-
	LA	35217	65.9%	92.1%	42151
LoHi	Min-Min	558123	71.2%	61.0%	-
	RC	521603	58.6%	79.1%	-
	LA	539132	49.1%	91.2%	39137
HiHi	Min-Min	1667186	71.1%	61.2%	-
	RC	1551127	58.8%	79.5%	-
	LA	1630903	49.0%	90.6%	35604

Table 3 presents results for consistent environments. LA results are close to RC and Min-Min, but are not better than them. A comparison

between LA, RC and Min-Min over makespan is shown for three different environments in Figure 5. The values are normalized to makespan derived from LA. It clearly shows that for inconsistent environments, which are highly heterogeneous, our proposed algorithm improves best existing algorithm by 15 percent. For less heterogeneous environments such as semi-consistent and consistent environments, RC and Min-Min result in slightly better performance. For more experiments and analysis, reader may refer to [12].

**Table 2. Comparison: consistent environment**

		Makespan	Matching proximity	System utilization	Iteration
LoLo	Min-Min	18905	45.2%	83.5%	-
	RC	18090	42.1%	93.6%	-
	LA	19461	40.0%	91.5%	41306
HiLo	Min-Min	56673	45.2%	83.4%	-
	RC	54204	42.0%	93.8%	-
	LA	58168	40.1%	91.5%	43955
LoHi	Min-Min	981799	36.5%	67.3%	-
	RC	1001843	26.7%	90.3%	-
	LA	1139402	23.3%	90.8%	50531
HiHi	Min-Min	2948916	36.5%	67.2%	-
	RC	2997984	26.6%	90.6%	-
	LA	3419437	23.3%	90.6%	52199

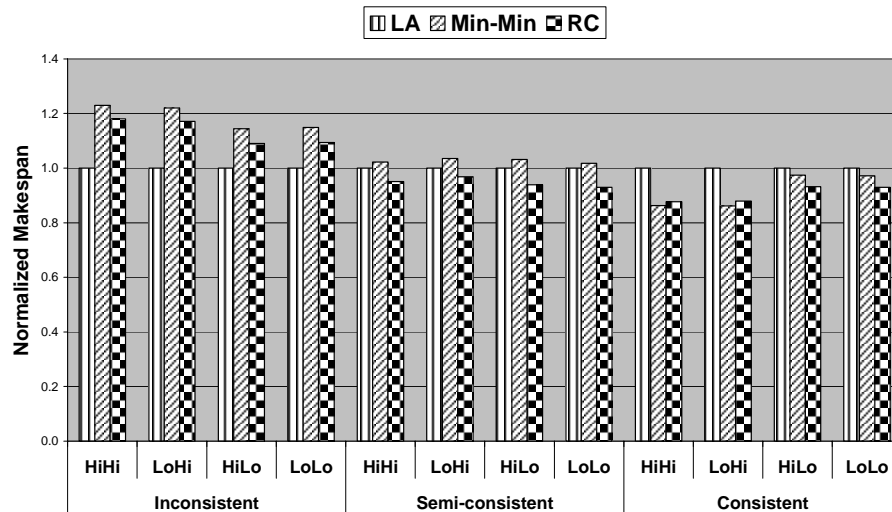


Figure 5. Comparison over twelve heterogeneity combinations

## 7. Conclusion

This paper presents a mapping algorithm based on learning automata for a set of independent tasks over computational grid. The computational grid is modeled as a heterogeneous computing environment. The objective of the proposed algorithm is to assign tasks to machines in a way to minimize *makespan*. Makespan is dependent on two criteria, matching proximity and system utilization to minimize the makespan. The algorithm uses these two measures to guide learning automata towards a good mapping. Through experiments, we showed that for high heterogeneous environment, i.e. inconsistent environments, the proposed algorithm improves two best existing algorithms.

## 8. References

- [1]. A. L. Rosenberg, Optimal Scheduling for Cycle-stealing in a Network of Workstations with a Bag-of-tasks Workload, *IEEE Transaction on Parallel Distributed Systems*, Vol. 13, No. 2, 2002, pp. 179-191.
- [2]. H. Casanova, T.M. Bartol, J. Stiles, F. Berman, Distributing MCell simulations on the grid, *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, 2001, pp. 243-257.
- [3]. S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M. Su, C. Kesselman, S. Young, M. Ellisman, Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience, *IEEE Proceedings of the Ninth Heterogeneous Computing Workshop*, 2000, pp. 241-252.
- [4]. M. Macheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems, *Journal of*

*Parallel and Distributed Computing*, Vol. 59, No. 2, 1999, pp. 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, *Journal of Parallel and Distributed Computing*, Vol 61, 2001, pp. 810-837.

[6]. T. D. Braun, H. J. Siegel, et al., Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems, *Proceedings of the 17<sup>th</sup> IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 330-335.

[7]. Min-You Wu and Wei Shu, A High-Performance Mapping Algorithm for Heterogeneous Computing Systems, *Proceedings of 15<sup>th</sup> International 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, pp. 279-310.

[10]. H. Chen and M. Maheswaran, Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems, *Proceedings of International 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, *Journal of ACM*, Vol. 24, No. 2, 1977, pp. 280-289.

[12]. S. Ghanbari and M. R. Meybodi, Various Mapping Algorithms for Heterogeneous Computational Grid: Learning Automata Approach, *Technical Report, Computer Engineering Department and Information Technology*, Tehran, Iran, 2004.