A Batch-mode Mapping Algorithm in Highly Heterogeneous Computational Grids using Learning Automata

S. Ghanbari[1] and M. R. Meybodi[2]
Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University
saeed_ghanbari@yahoo.com

Abstract

The computational grid is a new paradigm in parallel and distributed computing systems for realizing a virtual supercomputer over idle resources available in a wide area network like the Internet. Computational Grids are characterized for exploiting highly heterogeneous resources; so, one of the main concerns in developing computational grids is how to effectively map tasks onto heterogeneous resources in order to gain high utilization. Two approaches for mapping the tasks exist, online mode and batch mode. In batch-mode at any mapping event, a batch of tasks is mapped, whereas in online mode only one task is mapped. In this paper, two batch-mode 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 three best existing mapping algorithms when the heterogeneity of the environment is very high.

Keywords: Computational grid, Metatask, Mapping, Learning automata, Heterogeneous computing

1. Introduction

Computational grid is a new paradigm in parallel and distributed computing systems for realizing a virtual supercomputer over idle resources available in a wide area network like the Internet. 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 scheduling.

A grid is made of resources with diverse specifications, which makes the computing environment highly heterogeneous. On the other hand, due to dynamically fluctuating delays, changing quality of services, and unpredictable behavior, a reliable communication among tasks is nearly impossible. Regarding to these obstacles, the scheduling of applications on computational grid 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]). A 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].

[1] MS Student, Computer Engineering Department, Amirkabir University
[2] Professor of Computer Engineering Department, Amirkabir University

In general, two approaches for mapping the tasks exist, *online mode* and *batch mode*. In batch mode, at any mapping event a batch of tasks are mapped, whereas in online mode only one task is mapped. In this paper, we present two batch-mode algorithms based on learning automata for mapping metatask over HC. Through computer simulation 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 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 based algorithms. In Section 6, experimental results are discussed, and section 7 is the conclusion.

2. Related Works

As mentioned in the previous section, 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[5,4], and Relative Cost (RC) [7] . Experiments 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. It is shown that in order to minimize the makespan, matching and system utilization should be maximized, and an ideal algorithm should satisfy both criteria simultaneously.

Learning automata have been always a tempting method for addressing implication of scheduling in distributed computing systems. Stonkovic et. al. [12], a distributed scheduling framework, needs 2^N learning automata for N machines, which is not applicable for large number of machines. In [13] cooperative behavior of two learning automata is investigated. [14] proposes a learning automata model for optimizing multiple costs in a heterogeneous computing system, where an application is modeled as a directed graph and suite of machines are modeled as an undirected graph.

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. 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.

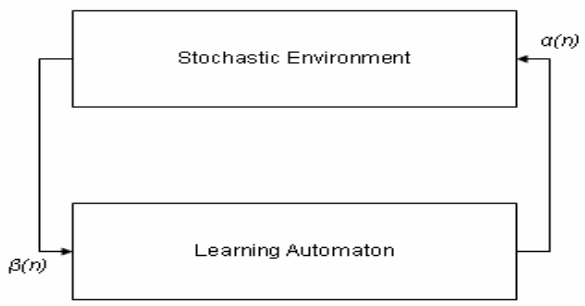

Figure 1. The interaction between learning automata and environment

A VSLA is a quintuple $< \alpha, \beta, p, T(\alpha,\beta,p) >$, where α, β, 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 refer 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 $\beta(i) \epsilon [0,1]$. A general linear schema for updating action probabilities can be represented as follows. Let action i be performed then

$$p_j(n+1) = p_j(n) + \beta(n)[b/(r-1) - bp_j(n) - [1-\beta(n)]ap_j(n) \quad \forall j \quad j \neq i$$
$$p_i(n+1) = p_i(n) - \beta(n)bp_i(n) + [1-\beta(n)]a[1-p_i(n)] \qquad (1)$$

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 $L_{R\epsilon P}$. 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. 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 automata, and a 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. 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.

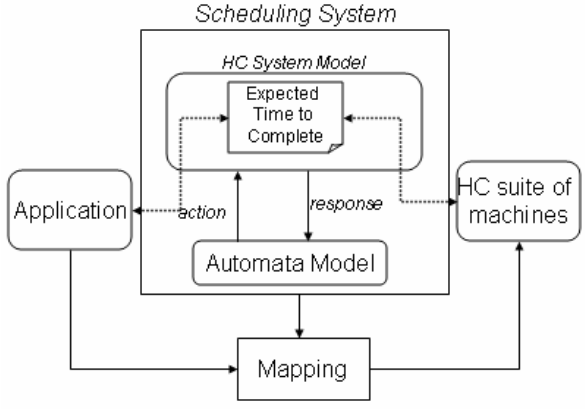

Figure 2-Model of the Grid

We define $\psi^{(n)}(i)=j$ as a general mapping from the task domain $i=1,...\tau$ to the machine domain $j=1,...,\mu$ 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:

$$\theta^{(n)}(j) = \sum ETC(k,j), j = \psi^{(n)}(k)\ 1 \le k \le \tau \quad \textbf{(2)}$$

The maximum value 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, and 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, $1 \le i \le \tau$, $a(i)=m_1,m_2,...,m_\mu$(m_i is the i^{th} 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}.

At each iteration, each learning automaton selects a machine, and then, makespan and load of each machine is computed. This loop continues till one of termination conditions is fulfilled.

To determine the goodness of an action taken by an automaton, we have proposed a class of algorithms which evaluate the reward for each learning automaton by considering difference in makespan and load of chosen machine at each iteration with their values in the previous iteration. Depending on the value of makespan being less or greater than a threshold, the algorithms apply two reward policies. The main reason behind using a predefined threshold is to guide the learning automata to find a solution not worse than the threshold. We refer to this class of algorithms as *Threshold-based Deterministic* (TD) algorithms.

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, and makespan being below or above the threshold, 18 different states are possible. To determine the $\beta^{(n)}(i)$, we associate a reward value to each nine possible state, which determines the goodness of the chosen action. A generalized formulation for TD algorithms is presented in the Eq.3.

$$\beta^n(i) = \begin{cases} 1-(f_b(T^{n-1},T^n)\beta_T + f_b(\theta^{n-1}(\psi^{n-1}(i)),\theta^n(\psi^n(i)))\beta_L) & if\ T^n < \Omega \\ 1-(f_o(T^{n-1},T^n)\beta_T + f_o(\theta^{n-1}(\psi^{n-1}(i)),\theta^n(\psi^n(i)))\beta_L) & if\ T^n \ge \Omega \end{cases} \quad \textbf{(3)}$$

where T^n stands for *makespan* and Ω is the predefined threshold. β_T represents rewarding value as far as *makespan* is concerned, and β_L represents rewarding value as far as load of chosen machine is concerned, and $\beta_T+\beta_L=1$. The greater the β_T the more impact of variation in makespan in evaluation of environment response will be. Similarly, the greater the β_L the more impact of variation in load of chosen machine in evaluation of environment response will be. Functions f_o and f_b determine the rewarding policy when the makespan is *over* and *below* the predefined threshold, respectively. Based on defining f_o and f_b, various TD algorithms could be proposed. In this paper, two algorithms are proposed and analyzed, which are called TD1 and TD2. These two proposed algorithm interpret the environment as a Q-model.

Algorithm TD1 policy of rewarding is as follows: if the makespan is above the threshold, decrease state (in makespan or load of machine in contrast to previous iteration) is associated with full reward, remaining unchanged state is associated with half reward and increase state is associated with no reward. If the makespan is below the threshold, decrease and remaining unchanged states are associated with full reward, while increase state is associated with no reward. Hence, f_o and f_b for algorithm TD1 are defined as presented in Eq.4 and Eq.5.

$$f_o(X,Y) = \begin{cases} 0 & X < Y \\ \frac{1}{2} & X = Y \\ 1 & X > Y \end{cases} \quad \textbf{(4)}$$

$$f_b(X,Y) = \begin{cases} 0 & X < Y \\ 1 & X \ge Y \end{cases} \quad \textbf{(5)}$$

In algorithm TD2 in contrast to algorithm TD1, remaining unchanged and increase states are associated with no reward in the case of makespan being over the threshold. In the case of makespan being below the threshold, algorithm TD2 associates rewards to the possible states the same way as algorithm TD1. Hence, f_o and f_b for algorithm TD2 are defined as presented in Eq.4 and Eq.6.

$$f_o(X,Y) = \begin{cases} 0 & X \le Y \\ 1 & X > Y \end{cases} \quad \textbf{(6)}$$

As stated previously, threshold is a measure to impose a constraint to learning automata in order to find a solution not worse than a predefined level. We choose makespan resulted from best existing algorithms as the threshold.

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 $1 \le i \le \tau$. 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, $1 \le j \le \mu$. 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 $1 \le j \le \mu$. (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.

The results reported here are averaged over 50 trials. All experiments are done for 200 tasks and 20 machines. The makespan for each experiment is normalized with respect to the benchmark heuristic, which is RC. The learning automata model used is L_{RI} with $a=0.001$. β_L and β_T set to 0.75 and 0.25, respectively. The threshold for finding a mapping for each metatask is set to the makespan resulted from running algorithm Min-Min on the metatask.

Termination condition is met when, no change in makespan is made for 1500 consecutive iterations, or number of iterations exceeds 500000.

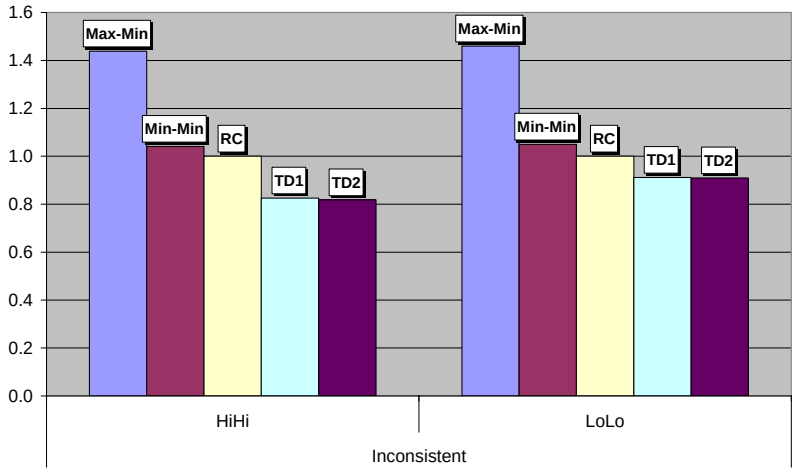

Figure 3. Makespan for inconsistent environment

In Figure 3, two proposed algorithms are compared with Max-Min, Min-Min and RC in term of normalized makespan for inconsistent heterogeneity. It can be noted that both proposed algorithms outperform both RC and Min-Min. For high machine and task heterogeneity, makespan resulted by algorithms TD1 and TD2 are about 20 percent less than the makespan resulted from RC. For low machine and task heterogeneity, the proposed algorithms result in makespan with about 9.5 percent less than RC. It should be noted that, according to [5][7], RC always performs better than all existing algorithms including GA, Simulated annealing, and Min-Min. Therefore outperforming RC implies outperforming all reported algorithms in the literature.

Figure 4 compares the proposed algorithms with the Max-Min, Min-Min and RC in term of normalized makespan for semi-consistent heterogeneity. For high task and machine heterogeneity, both algorithms TD1 and TD2 yield in makespan 5 percent lower than RC. For low task and machine heterogeneity, algorithms TD1 and TD2 results are very close to RC, yet outperform Min-Min.

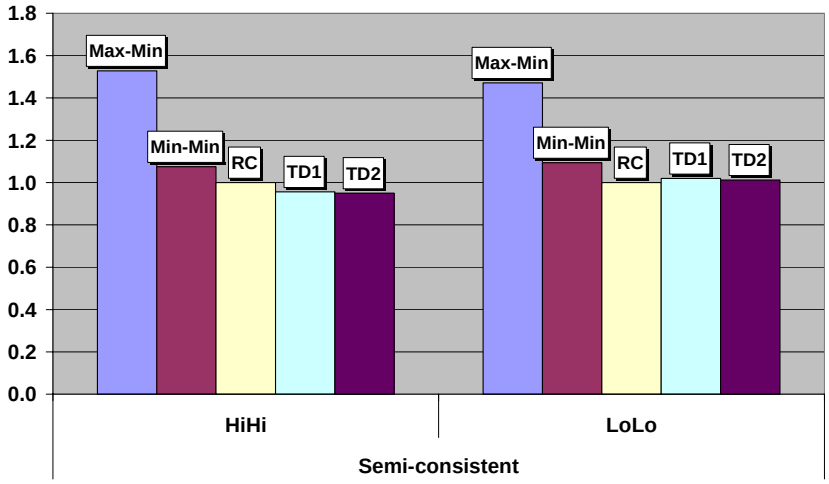

Figure 4. Makespan for semi-consistent environment

Figure 5 compares the proposed algorithms for consistent environment. For consistent environments with low task and machine heterogeneity, it can be seen that proposed algorithms perform slightly better than Min-Min; however, they fail to improve RC. Comparing the presented results indicates the fact that proposed algorithms perform significantly better than both RC and Min-Min for inconsistent environments, while they fail 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 perform better in environments with higher level of heterogeneity.

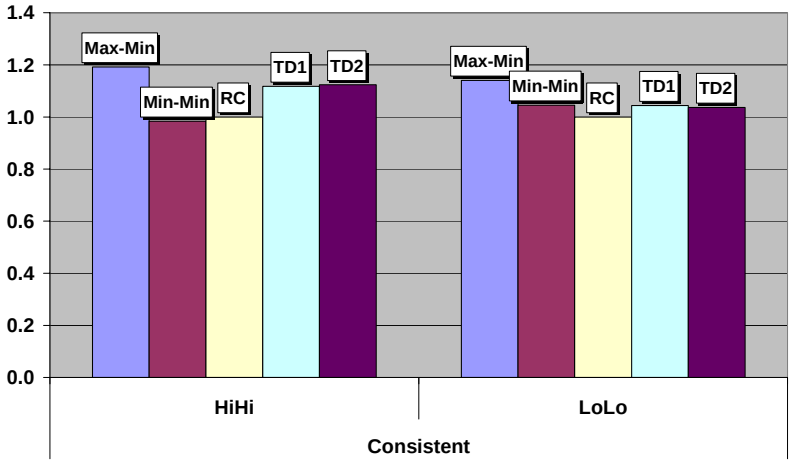

Figure 5. Makespan for consistent environment

The other important issue to consider is computational cost of finding a mapping using each algorithm. From **Figure 6** which shows number of iterations needed for finding a mapping in different environments, it can be inferred that for lower level of heterogeneity, proposed algorithms require more iterations. As an justification, in the consistent environment, the first, second, third,…, and μ^{th} fastest machines for all tasks are the same. To avoid pouring all tasks on one machine, each learning automaton must select a machine which is not the fastest for the assigned task. Therefore, finding an optimum mapping requires cooperation among learning automata which our proposed model lacks it. This accounts for poorer results in consistent environment. Fortunately, computational grids and, in general, heterogeneous computing systems are best characterized as inconsistent and semi-consistent environments.

7. Conclusion

This paper presented two batch-mode algorithms based on learning automata for mapping a set of independent tasks over computational grid. The computational grid was modeled as a heterogeneous computing system and the objective of the proposed algorithms was to assign independent tasks to machines in a way to minimize *makespan*. Through experiments, we showed that for highly heterogeneous environments, i.e. inconsistent environments, the proposed algorithms outperform best existing mapping algorithms.

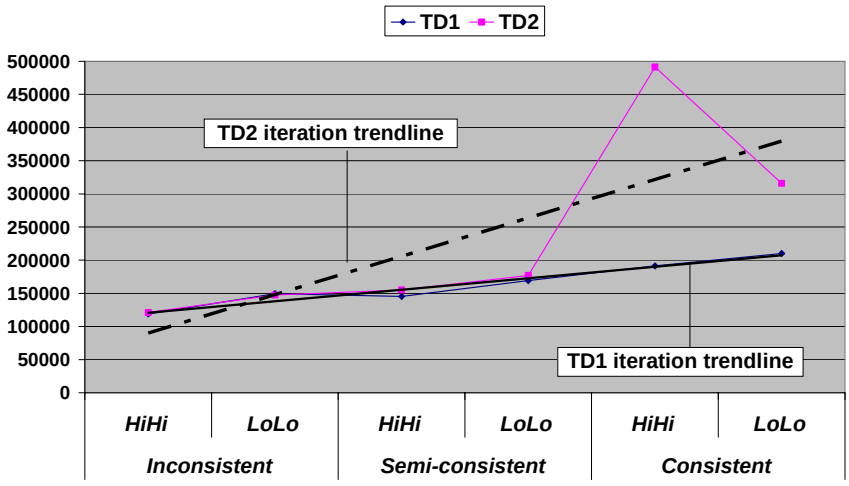

Figure 6. Iteration needed in different environments.

References

[1] L. Rosenberg, "Optimal scheduling for cycle-stealing in a network of workstations with a bag-of-tasks workload," *IEEE Trans. Parallel Distributed Systems*, Vol.13, No.2, pp. 179–191, 2002.

[2] H. Casanova, T.M. Bartol, J. Stiles, F. Berman, "Distributing MCell simulations on the grid," *Int. J. High Performance Computing*. *Application*, Vol.15, No. 3, pp. 243–257, 2001.

[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," *Proc. 9th IEEE Heterogeneous Computing Workshop*, pp. 241–252, 2000.

[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," *J. Parallel Distributed Computing*, Vol.59, No.2, pp. 107–131,1999.

[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,* Vol. 61, No.6, pp. 810–837, 2001.

[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*, pp. 330–335, 1998.

[7] Min-You Wu and Wei Shu, "A high-performance mapping algorithm for heterogeneous computing systems," *Proc. of 15th Int. Parallel and Distributed Processing Symposium*, pp.74, 2001.

[8] K. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction*, Prentice Hall, 1989.

[9] F. Berman, "High-performance schedulers in the grid", in *Blueprint for a New Computing Infrastructure*, *I. Foster and C. Kesselman(eds)*, Morgan Kaufmann, pp. 279–309, 1999.

[10] H. Chen and M. Maheswaran, "Distributed dynamic scheduling of composite tasks on grid computing systems," *Proc. Int'l Parallel and Distributed Processing Symposium*, 2002.

[11] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *J. ACM*, Vol.24, No. 2, pp.280–289, 1977.

[12] R Mirchandaney and J. A. Stankovic. "Using stochastic learning automate for job scheduling in distributed processing systems," *J. Parallel and Distributed Computing*, Vol.3, pp. 527–552, 1986.

[13] A. Glockner and J. Pasquale, "Coadaptive behavior in a simple distributed job scheduling system," *IEEE Trans. on Systems, Man and Cybernetics*, Vol.23, No.3, pp. 902–907, 1993.

[14] R. D. Venkataranna and N. Ranganathan, "Multiple cost optimization for task assignment in heterogeneous computing systems using learning automata," *IEEE Trans.*, pp. 137–145, 1999.