On-line Mapping Algorithms in Highly Heterogeneous Computational Grids: A Learning Automata Approach

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

Abstract

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 are mapped, whereas in online mode only one task is mapped. In this paper, four on-line mode mapping algorithms based on learning automata are introduced. To show the effectiveness of the proposed algorithms, computer simulation has been conducted. The results of experiments show that the proposed algorithms outperform two best existing mapping algorithms when machine heterogeneity high.

Keywords: Computational grid, Online mapping, Learning automata, Heterogeneous computing

1. Introduction

Computational grid is a new paradigm in distributed computing which aims to realize a large-scale high performance computing environment over geographically distributed resources. Computational grid enables sharing, selection, and aggregation of highly heterogeneous resources for solving large scale problems in fields of 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 crucial issue to meeting the computational requirements on grid is resource allocation.

Resources on grid are typically shared and undedicated so that the contention made by various applications results in dynamically fluctuating delays, changing quality of services, and unpredictable behavior, which further complicates the scheduling. On the other hand, architectures of machines available in a grid are very diverse in specification so as to meet different application requirements; therefore, the extent to which a given task can exploit a given architectural feature depends on how well the task's computational requirements match the machine's advanced capabilities. In brief, grid resource management faces to two major problems; one is matching computational needs to appropriate resources, and the other is exploiting resources over highly dynamic environment.

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 onto machines in a *heterogeneous computing* (HC) system has been shown to be NP-complete [1]. Mapping heuristics can be classified into two categories, *on-line mode* and *batch mode* [5]. In the *on-line mode*, a task is mapped onto a machine as soon as it arrives at the scheduler. In the *batch mode*, tasks are collected into a set, called *metatask*, which is further examined for mapping at prescheduled times.

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

Learning automata are adaptive decision-making devices operating on unknown random environments. They have proved capable of finding proper solutions for NP-complete problems. In this paper, the main concern is to address the implications imposed by heterogeneity of applications and resources in computational grid. We have proposed four on-line mapping algorithms based on learning automata. Simulation studies are performed to compare the results of proposed algorithms with two best existing heuristics.

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 studied grid and the definitions used in later sections. Section 5 introduces the proposed algorithms. In Section 6, experimental results are discussed and section 7 is conclusion.

2. Related Works

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[3], fractals calculations (such as Mandelbrot), and image manipulation applications (such as tomographic reconstruction[4]).

As mentioned in the previous section, existing mapping algorithms can be categorized into two classes[5]: 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[5]. The objective of most mapping algorithms is to minimize *makespan*; the time needed for completing the execution of a metatask. Minimizing *makespan* yields to higher throughput.

Five on-line mode heuristics are reported in the literature[5]. These are (i) minimum completion time(MCT), (ii) minimum execution time(MET), (iii) switching algorithm(SA), (iv) k-percent best(KPB), and (v) opportunistic load balancing(OLB).

The minimum completion time (MCT) heuristic assigns each task to the machine which results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The MCT heuristic is a variant of the fast-greedy heuristic from SmartNet[9].

The minimum execution time (MET) heuristic assigns each task to the machine that performs that task's computation in the least amount of execution time (this heuristic is also known as LBA (limited best assignment) [13] and UDA (user directed assignment)[9]). This heuristic, in contrast to MCT, does not consider machine ready times. This heuristic can cause a severe imbalance in load across the machines. The advantages of this method are that it gives each task to the machine that performs it in the least amount of execution time, and the heuristic is very simple.

The switching algorithm (SA) is motivated by the following observations. The MET heuristic can potentially create load imbalance across machines by assigning many more tasks to some machines than to others, whereas the MCT heuristic tries to balance the load by assigning tasks for earliest completion time. If the tasks are arriving in a random mix, it is possible to use the MET at the expense of load balance until a given threshold and then use the MCT to smooth the load across the machines. The SA heuristic uses the MCT and MET heuristics in a cyclic fashion depending on the load distribution across the machines. The purpose is to have a heuristic with the desirable properties of both the MCT and the MET.

The *k*-percent best (KPB) heuristic considers only a subset of machines while mapping a task. The subset is formed by picking the $m\times(k/100)$ best machines based on the execution times for the task, where $100/m\leq k\leq 100$. The task is assigned to a machine that provides the earliest completion time in the subset. If $k=100$, then the KPB heuristic is reduced to the MCT heuristic. If $k=100/m$, then the KPB heuristic is reduced to the MET heuristic. A "good" value of k maps a task to a machine only within a subset formed from computationally superior machines. The purpose is not so much to match the current task to a computationally well-matched machine as it is to avoid putting the current task onto a machine which might be more suitable for some yet-to-arrive tasks. It should be noted that while both the KPB and the SA combine elements of the MCT and the MET in their operation, it is only in the KPB that each task assignment attempts to optimize objectives of the MCT and the MET simultaneously.

The opportunistic load balancing (OLB) heuristic assigns a task to the machine that becomes ready next, without considering the execution time of the task onto that machine. If multiple machines become ready at the same time, then one machine is arbitrarily chosen. In [5] it is shown that KPB and MCT result in better makespan than other mentioned heuristics.

Learning automata proved an efficient tool for addressing implication of scheduling in distributed computing. Stonkovic et. al. [10], a distributed scheduling framework, needs 2^N learning automata for N machines, which is not applicable for large number of machines. In [11] cooperative behavior of two learning automata is investigated. [12] 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].

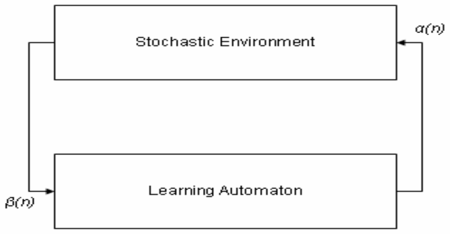

Figure 1. The interaction between learning automata and environment

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

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

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 $L_{R\varepsilon P}$.

A fixed structure learning automaton is represented by a quintuple < *α, Φ, β, F, G* >, where $\Phi=\{\varphi_1, \ldots, \varphi_\sigma\}$ is the set of internal states, $F: \Phi \times \beta \rightarrow \Phi$ is a map called the state transition map which defines the transition of the state of the automaton on receiving an input, and $G: \Phi \rightarrow \alpha$ is the output map and determines the action taken by the automaton. On the basis of the response *β(v)* at the time n, the state of automaton is updated and a new action is chosen at the time n+1. Based on state transition map and output map, different fixed structured learning automata can be defined. $L_{2N,2}$, $G_{2N,2}$, Krynsky and Krylov are fixed structure learning automata used in this paper. For more information about learning automata the reader may refer to [8].

4. Simulation Model

This section presents a general model of computational grid. The application and heterogeneous 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.

Tasks arrive as a Poisson stream, where t_i denotes the arrival time of task s_i. Each task is put into the queue of a machine, and executed in a First-Come-First-Served basis.

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

$$\theta^{(t)}(j)=\sum ETC(k,j)-t,\ j=\psi(s_k)\ \ for\ t_k\le t \qquad \textbf{(2)}$$

The maximum value of $\theta^{(t)}(j)$, over $1\le j\le\mu$, at time t is referred to as *makespan* and denoted by $\theta^{(t)}_{max}$.

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

$$\psi_{\min}(i)=j \text{ such that } ETC(i,j)=\min_{1\le q\le\mu} ETC(i,q) \qquad \textbf{(3)}$$

Thus, *matching proximity* at time t is defined as follows:

$$\eta(t)=\frac{\sum_{0\le t_i\le t} ETC\ \ (i,\psi_{\min}\ (i))}{\sum_{0\le t_i\le t} ETC\ \ (i,\psi\ (i))} \qquad \textbf{(4)}$$

where $0<\eta(t)\le 1$. When $\eta(t)=1$, we have the ideal matching.

Load balance can be measured by *system utilization*:

$$\delta(t)=\frac{\sum_{0\le t_i\le t} ETC\ (i,\Psi(i))}{\mu\times\theta^{(t)}_{\max}} \qquad \textbf{(5)}$$

When the system is completely balanced, $\delta(t)=1$; otherwise $\delta(t)<1$. Thus, the makespan of a mapping can be expressed as:

$$\theta^{(t)}_{\max}=\frac{\sum_{1\le t_i\le t}\psi_{\min}(i)/\mu}{\eta(t)\times\delta(t)} \qquad \textbf{(6)}$$

Therefore, to minimize the makespan, both $\eta(t)$ and $\delta(t)$ must be maximized.

5. Proposed Learning Automata Model

As depicted in figure 2, the proposed algorithms use a learning automaton with l actions, each associated with a heuristic. Each heuristic is supposed to map the arriving task in a way that a certain criterion is improved. The main idea is to combine preexisting mapping heuristics to have mappings optimized in different criteria which are necessary for a high quality mapping.

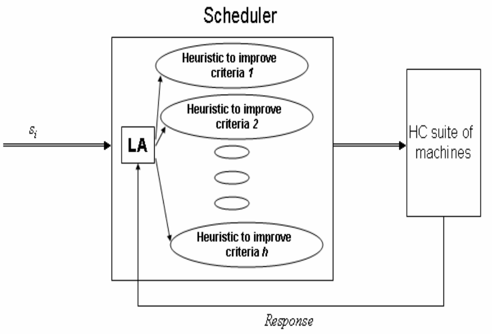
Scheduler
Heuristic to improve criteria 1
Heuristic to improve criteria 2
Heuristic to improve criteria h
s_i
LA
HC suite of machines
Response

Figure 2. Learning automata model for on-line mapping algorithms

$C=\{c_1,c_2,\ldots,c_h\}$ is the set of criteria, and the automaton is represented by a triple $(a, \beta(i), A)$, where $a=l_1,l_2,\ldots, l_h$ (l_i is the i^{th} heuristic for improving criteria i). $\beta(i)$ represents the environment response after mapping task s_i. Figure 3 shows the general procedure of proposed algorithms.

```
For each $s_i$
Begin
 A.Select-mapping-Policy()
 $\psi(i)$=Select-machine according to mapping policy
 Evaluate $\beta(i)$ according to selected mapping policy
 A.Update($\beta(i)$)
End
```

Figure 3. General procedure for proposed algorithms

Two criteria are considered in proposed algorithms: *matching proximity* and *system utilization*. To improve matching proximity, heuristic MET is used, and to improve system utilization, heuristic MCT is used. Thus, the automaton has two actions associated with heuristics MCT and MET.

Depending on evaluating the environment response based on absolute or relative value of criteria, and on using variable or fixed structure learning automaton, four different algorithms are proposed.

Algorithms LMFS and LMVS

Algorithms LMFS and LMVS evaluate the environment response based on absolute value of system utilization and matching proximity. LMFS uses a fixed structured learning automaton, while LMVS uses a variable structured learning automaton.

After selecting an action and execution of the related heuristic for task s_i, $u(s_i)$ which is a linear combination of matching proximity and system utilization is evaluated as shown in Eq.7.

$$u(s_i)=(1-e^{-\frac{1}{2}\left(\frac{\eta(s_i)-1}{0.5}\right)^2})\lambda_\eta+(1-e^{-\frac{1}{2}\left(\frac{\delta(s_i)-1}{0.5}\right)^2})\lambda_\delta \qquad \textbf{(7)}$$

where $\lambda_\eta + \lambda_\delta = 1$. λ_η and λ_δ, , are weightages associated with matching proximity and system utilization. These values determine the extent of impact of each criterion in evaluating the environment response.

Algorithm LMVS interprets the environment as an S-Model and evaluates the environment response as $\beta(i)=u(s_i)$. Algorithm LMFS interprets the environment as a P-Model. It evaluates the environment response as $\beta(i)=I(u(s_i))$, where $I(p)$ is an *Indicator* function which returns 1 with probability of p, and 0 with probability of $1-p$.

Algorithms ILMFS and ILMVS

Algorithms ILMFS and ILMVS, despite two previously illustrated algorithms, consider improvement in matching proximity and system utilization for evaluating environment response. ILMFS uses a fixed structured learning automaton, while ILMVS uses a variable structured learning automaton.

In these algorithms, if a criterion (matching proximity or system utilization) improves after mapping the newly arriving task, the automaton receives full reward according to that criterion. But, in the case of decline in the criterion, the automaton receives penalty proportional to the decline. We define two quantities η' and δ', which are evaluated as shown in Eq.8 and Eq.9, respectively.

$$\eta'(s_i)=\begin{cases}0 & \Delta\eta(s_i)>0\\ 1-e^{-\frac{1}{2}\left(\frac{\Delta\eta(s_i)-0.001}{0.005}\right)^2} & otherwise\end{cases} \qquad \textbf{(8)}$$

$$\delta'(s_i)=\begin{cases}0 & \Delta\delta(s_i)>0\\ 1-e^{-\frac{1}{2}\left(\frac{\Delta\delta(s_i)-0.001}{0.005}\right)^2} & otherwise\end{cases} \qquad \textbf{(9)}$$

$\Delta\eta(s_i)$ is the difference between matching proximity of mappings of previous task and current task: $\Delta\eta(s_i)=\eta(t_i)-\eta(t_{i-1})$. Similarly, $\Delta\delta(s_i)$ is the difference between system utilization of mappings of previous task and current task: $\Delta\delta(s_i)=\delta(t_i)-\delta(t_{i-1})$. $u(s_i)$, used to evaluate environment response, is defined as a linear combination of matching proximity and system utilization:

$$u(s_i)=\eta'(s_i)\lambda_\eta+\delta'(s_i)\lambda_\delta \qquad \textbf{(10)}$$

where $\lambda_\eta + \lambda_\delta = 1$. λ_η and λ_δ are weightages associated with matching proximity and system utilization. These values determine the extent of impact of each criterion in evaluating the environment response.

Interpreting the environment as an S-Model, algorithm ILMVS evaluates the environment response as $\beta(i)=u(s_i)$. Algorithm ILMFS evaluates the environment response as $\beta(i)=I(u(s_i))$. It interprets the environment as a P-Model.

6. Experiments

In this section the proposed algorithms are tested and compared with two best existing algorithms : Min-Min and RC. For 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.

Low task, Low task machine heterogeneity (LoLo) models a grid with little diversity in applications and machines. It represents the least heterogeneous environment. Real computational grids are best modeled as Low task, High machine heterogeneity (LoHi) because computational grids are usually used for executing limited number of highly intensive computational applications on machines with diverse architectural specification. High task, High machine heterogeneity (HiHi) models the most heterogeneous environment, where there are applications with diverse computational requirements on highly different machines. High task, Low machine heterogeneity (HiLo) represents a situation that the applications are very diverse in computational demands, but machines are not diverse in architectural specification. Computational grids are less expected to be modeled as HiLo. The experimental results are provided for all four possible task/machine heterogeneities.

Being the best on-line mapping heuristic, KPB is used as the benchmark heuristic, and with which proposed algorithms are compared. Comparison is made over θ^{τ}_{max} (the makespan resulted after assignment of last task). All reported results are normalized with respect to the result of the benchmark heuristic. For each experiment, task arrival rate is chosen to allow 90% of tasks have completed by the arrival of last task in the set when benchmark heuristic is used for mapping. The results reported here are averaged over 50 trials. All experiments are done for task set of size 2000, and over 20 machines.

Algorithms LMFS and ILMFS are tested using $L_{2N,2}$, $G_{2N,2}$, Krinsky, and Krylov, with memory depth of 2 for LMFS, and 3 for ILMFS. Algorithms LMVS and ILMVS are tested using L_{RI} with $a=0.5$, $L_{R\epsilon P}$ with $a=0.1$ and $b=0.01$, and L_{RP} with $a=b=0.1$.

Figure 4 compares the normalized makespan of different proposed algorithms for HiHi heterogeneity. Except LMVS and ILMVS with L_{RI}, all proposed algorithms outperform KPB and MCT. Results of algorithm LMFS are better than other proposed algorithm which at the best case, it improves KPB by 20% when using $L_{2N,2}$ and Krylov. In figure 5 results of proposed algorithms are compared for LoHi heterogeneity. The results show that by choosing a proper automaton, each proposed algorithm outperforms KPB.

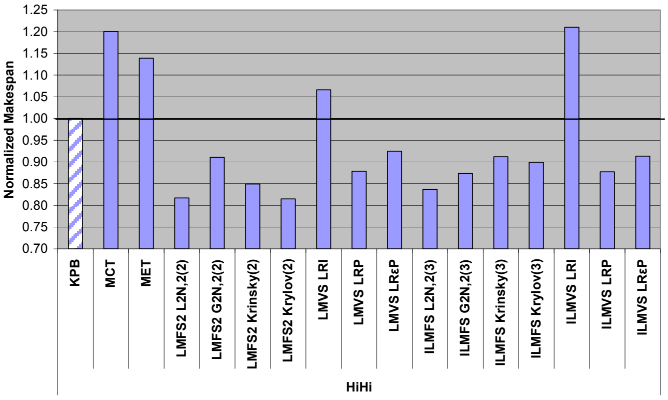

Figure 4. Makespan of the proposed algorithms for HiHi heterogeneity

Results depicted in figure 6 show that, at the best case for LoLo heterogeneity, proposed algorithms result in makespan 5% less than KPB. Results shown in figure 7 indicate that for HiLo heterogeneity, the proposed algorithms can barely outperform KPB.

In figure 8 and figure 9 makespan percentage of difference of proposed algorithms with KPB for different heterogeneities are shown. The results show that the proposed algorithms are best for high machine heterogeneity.

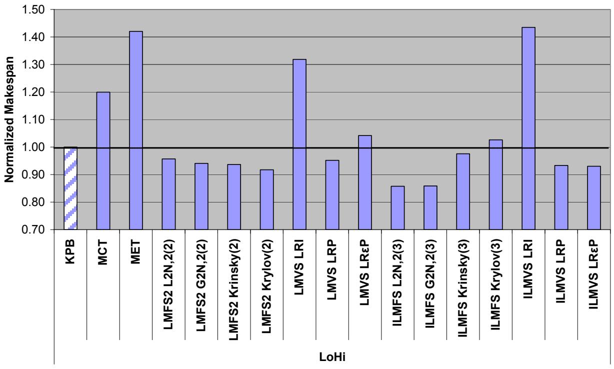

Figure 5. Makespan of the proposed algorithms for LoHi heterogeneity

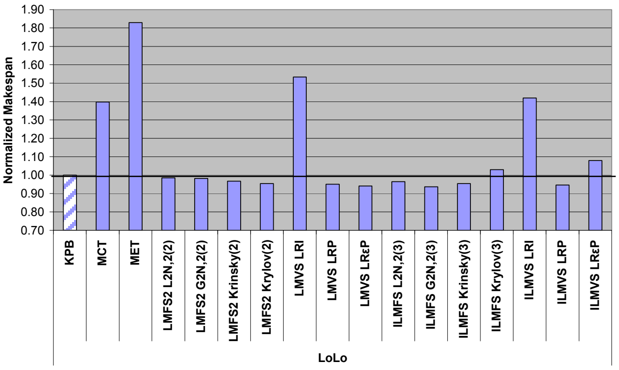

Figure 6. Makespan of the proposed algorithms for LoLo heterogeneity

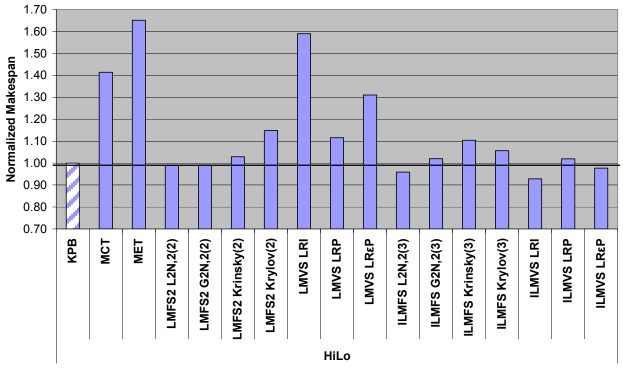

Figure 7. Makespan of the proposed algorithms for HiLo heterogeneity

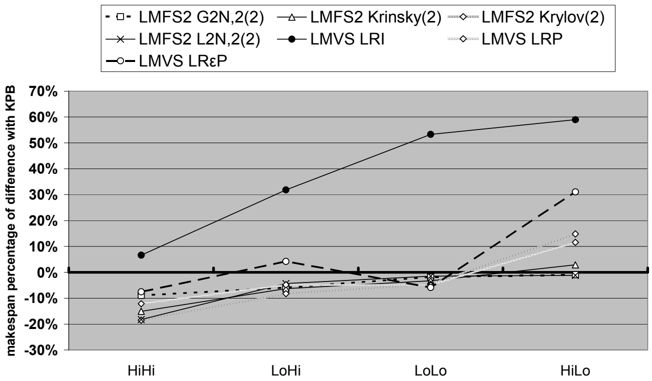

Figure 8. Makespan difference of LMFS and LMVS with KPB for different heterogeneities

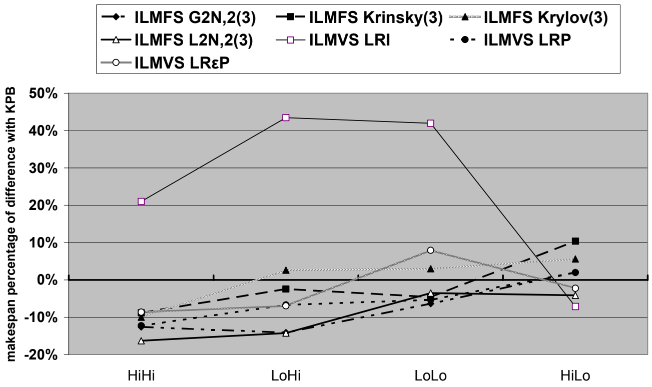

Figure 9. Makespan difference of ILMFS and ILMVS with KPB for different heterogeneities

7. Conclusion

This paper presented four online 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 high machine heterogeneity the proposed algorithms outperform best existing mapping algorithms.

References

[1] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *J. ACM,* 24(2), April 1977, pp.280–289.

[2] 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, pp. 330–335.

[3] H. Casanova, T.M. Bartol, J. Stiles, F. Berman, "Distributing MCell simulations on the grid," *Int. J. High Performance Computing*. *Application*, 15(3), 2001, pp. 243–257.

[4] 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*, 2000, pp. 241–252.

[5] 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*, 59(2), 1999 pp. 107–131.

[6] 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(6), 2001, pp. 810–837.

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

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

[9] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *Proc. 7th IEEE Heterogeneous Computing Workshop (HCW '98)*, 1998, pp. 184–199.

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

[11] A. Glockner and J. Pasquale, "Coadaptive behavior in a simple distributed job scheduling system," *IEEE Trans. on Systems, Man and Cybernetics*, 23(3), 1993, pp. 902–907.

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

[13] R. Amstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predications," *Proc. 7th IEEE Heterogeneous Computing Workshop*, 1998, pp. 79–87.