A Case-Based Recommender for Task Assignment in Heterogeneous Computing Systems

S. Ghanbari\textsuperscript{1} \quad M. R. Meybodi\textsuperscript{1} \quad K. Badie\textsuperscript{2}
\textsuperscript{1}Computer Engineering Department
Amirkabir University
Tehran, Iran
\textsuperscript{2}Information Group
Iran Telecommunication Research Center
Tehran, Iran

Abstract

Case-based reasoning (CBR) is a knowledge-based problem-solving technique, which is based on reuse of previous experiences. In this paper we propose a new model for static task assignment in heterogeneous computing system. The proposed model is a combination of the case based reasoning and the learning automata model. In this new model a learning automata model is used as adaptation mechanism which adapts previously experienced cases to the problem to be solved. The objective of the proposed model is to reduce the number of iterations required to find a semi-optimum solution. The application is modeled as a set of independent tasks and the heterogeneous computing system is modeled as a network of machines. Using computer simulation, it is shown that the combined model outperforms the model that only uses learning automata.

Keywords: Case-based Reasoning, Variable Structure Learning Automata, Heterogeneous Computing, Task Assignment

1 Introduction

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links, to perform different computationally intensive applications that have diverse computational requirements. The matching of the set of tasks (metatask) 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 known to be NP-complete[7].

Metatask is defined as a collection of independent tasks with no inter-task data dependencies. Metatasks occur in many situations. For example, all of the jobs submitted to a supercomputer center by different users would constitute a metatask. Another example of a metatask would be a group of image processing applications all operating on different images[5]. The mapping of the metatasks is being performed statically (i.e., off-line, or in a predictive manner). The goal of this mapping is to minimize the total execution time of the metatask. It is also assumed that each machine executes a single task at a time (i.e., no multitasking), in the order in which the tasks are assigned. The size of the metatask (i.e., the number of tasks to execute), and the number of machines in the HC suite are static and are known beforehand.

In the literature, varieties of mathematical formulations have been developed for the task assignment problem. Approaches to the problem based on graph theoretic techniques, simulated annealing[9], A* state space search[10], and genetic techniques[8] have been proposed. In [1] a learning automata model have been proposed [3]. The key feature of this model is its ability to optimize multiple cost metrics. The major problem with all the above mentioned models above is that for each task assignment, the models build the solution from the scratch and do not use their past experiences.

This paper presents a novel model for task assignment over a set of idle processors in a heterogeneous computing environment by means of case-based reasoning. Adaptation of stored cases and searching for new solutions is done by using learning automata. Using case-based reasoning approach enables the model to incorporate its former cases to find a suitable mapping of metatask to a set of processors in HC in a relatively short time. The proposed model is compared to the model reported in [1].

The rest of the paper is organized as follows. Section 2 introduces related works. In section 3, preceded by the description of HC system model and learning automata model, the proposed case-based reasoning model is introduced. Section 4 discusses experiments and results. The last section provides conclusions and a scope for future work.
2 Related Works

Case-based reasoning (CBR) is a knowledge-based problem-solving technique that is based on reuse of previous experience[4]. Unlike traditional knowledge-based techniques, which solve problems from scratch by reasoning with general knowledge, CBR focuses on specific problem-solving experience captured in cases that are collected in a case base. New problems are solved by retrieving cases that deal with previous similar problems. The solutions recorded in those similar cases are then adapted to become a solution for the new problem (this process is known as adaptation). Thus, the two basic hypotheses of CBR are similar problems have similar solutions, and reuse is more feasible than problem solving from scratch.

In recent years, case-based reasoning(CBR) which is a sort of analogical learning have shown high capabilities in different areas such as decision making, prediction, diagnosis, planning, quality/process control, decision support and information retrieval. However, few CBR applications are reported for task assignment and scheduling in distributed computing systems up to the composition of this paper. Other CBR applications in distributed computing are reported such as [18] which is a case-based expert system to help an organization assess its computing alternatives.

On the other hand, research on task assignment for scientific computations on homogeneous as well as heterogeneous systems has been extensively investigated in the literature[12][13]. There are variety of works done in this field such as: OLB(Opportunistic Load Balancing), MET (Minimum Execution Time)[14], MCT (Minimum Completion Time)[14], Min_min[14], Max_min[14],GA[14],A*[17], Learning automata[1].

3 Background

In this section we first introduce the HC system model and then give a brief introduction to the learning automata model which will be used as the case adaptation mechanism. These two subsections provide a basis for illustrating the proposed case-based reasoning model in detail in the next section.

3.1 HC System Model

This section presents a general model of the proposed framework for task assignment in the HC system. Figure 1 depicts a schematic representation of the framework. The environment consists of the heterogeneous suite of machines that are used to execute the application. The scheduling system consists of the proposed case-based recommender, HC system model and the learning automata model. These are used by the scheduler to assign the subtasks to different machines.

![Figure 1. Model of the proposed framework](image)

In the proposed model, some assumptions are made. Firstly, the metatask is assumed to be decomposed into multiple independent tasks. Secondly, the HC system is assumed to consist of a set of heterogeneous machines, which communicate by means of an underlying interconnection network. Thirdly, the expected execution time of the tasks on each machine in the HC suite is known a priori. These execution times can be obtained by task profiling and analytical benchmarking techniques[11].

The metatask $S$ contains a set of tasks $s_i$, the $i^{th}$ task in $S$ and $\tau = |S|$. The set of machines in the HC environment is given by $M$, where $m_i$ is the $i^{th}$ machine and $\mu = |M|$. $c$ denotes the cost metric defined for the metatask, i.e., the makespan. The assignment problem then corresponds to a mapping $\pi : S \rightarrow M$ from the set $S = \{s_i, 0 \leq i < \tau\}$ to the set $M = \{m_j, 0 \leq j < \mu\}$, such that the metric $c$ is optimized. A solution vector $X(n)$ of size $\tau$, gives an instance of the mapping $\pi$ at iteration $n$. The objective of the scheduling system is
to assign the tasks in the metatask to the processors in the HC, so that the defined cost criterion is optimized. The cost criterion is makespan, i.e., finish execution of the metatask as soon as possible.

The estimate of the expected execution time for each task on each machine is known prior to execution and contained within a $\tau \times \mu$ ETC (Expected Time to Compute) matrix. 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 $t_i$ and an arbitrary machine $m_j$, $ETC(t_i, m_j)$ is the estimated execution time of $t_i$ on $m_j$. The $ETC(t_i, m_j)$ entry could be assumed to include the time to move the executables and data associated with task $t_i$ from their known source to machine $m_j$. For cases when it is impossible to execute task $t_i$ on machine $m_j$ (e.g., if specialized hardware is needed), the value of $ETC(t_i, m_j)$ is set to infinity.

Machine availability time, $mat(mj)$, is the earliest time that machine $m_j$ can complete the execution of all tasks that have previously been assigned to it (based on the ETC entries for those tasks). The completion time for a new task $t_i$ on machine $m_j$, $ct(t_i, m_j)$, is the machine availability time for $m_j$ plus the execution time of task $t_i$ on machine $m_j$, i.e., $ct(t_i, m_j) = mat(m_j) + ETC(t_i, m_j)$. The performance criterion is the maximum value of $ct(t_i, m_j)$, for $0 \leq i < \tau$ and $0 < j < \mu$. The maximum $ct(t_i, m_j)$ value, over $0 \leq i < \tau$ and $0 < j < \mu$, is the metatask execution time, which is the makespan.

3.2 The Learning Automata Model

This section deals with the concept of learning automata (LA) first and then proceeds with a description of the learning automata model used as the adaptation mechanism.

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 2 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) [12]. In the following, the variable structure learning automata is described.

![Figure 2. The interaction between learning automata and environment](image)

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

If $\beta(n) = 0$,

$$p_i(n + 1) = p_i(n) + a[1 - p_i(n)]$$

$$p_j(n + 1) = (1 - a)p_j(n) \quad \forall j \neq i$$

If $\beta(n) = 1$,
\[ p_i(n + 1) = (1 - b) p_i(n) \]
\[ p_j(n + 1) = (b/s - 1) + (1 - b) p_j(n) \quad \forall j \neq i \]

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 \( L_{RP} \). For more information about learning automata the reader may refer to [3].

As mentioned before, learning automata is used as the case adaptation mechanism in the proposed task assignment problem. Figure 3 shows the schematic of the learning automata model. The model is constructed by associating every task \( s_i \) in the metatask with a variable structure learning automaton \( (\alpha^\beta, \beta^\gamma, A^\delta) \). Each learning automaton is represented by a 3-tuple \( (\alpha^\beta, \beta^\gamma, A^\delta) \).

Since the tasks can be assigned to any of the \( \mu \) machines (\( \mu \) number of machines), the action set of all learning automata are identical. It is assumed that the environment is P-model so the input set for learning automata is \( \{0,1\} \). Therefore for any task \( s_i \), \( 0 \leq i < \tau \) (\( \tau \) number of tasks), \( \alpha^\gamma = m_1, m_2, \ldots, m_{\mu-1} \) and \( \beta^\delta \in [0,1] \), where \( \beta^\delta \) equal to 0 indicates that the action taken by the automaton of task \( s_i \) is favorable to the system and \( \beta^\delta \) equal to 1 indicates an unfavorable response.

The output response of the HC system is assumed to be binary. If the value of makespan \( c(n) \) at iteration \( n \) is less than the value of makespan at iteration \( n - 1 \), then the output is favorable; otherwise it is unfavorable. That is, If \( c(n) \) is lower than \( c(n - 1) \) then \( R = 0 \) else \( R = 1 \), where \( R \) is the output response of the system with respect to \( c(n) \).

The next step in developing the model is to translate \( R \) into the input \( \beta(n) \) for each automaton. In [2] six heuristics are proposed for translating \( R \) into input \( \beta(n) \). In our model, a simple heuristic is used, i.e. \( \beta(n) = R \). We test the model for two different learning algorithms: \( L_{RI} \) and \( L_{RP} \). The general procedure for learning automata model is shown in Figure 4.

\[ \text{while(true)} \]
\[ \text{Begin} \]
\[ LA, \text{select action for all } 0 \leq i < \tau \]
\[ \text{Evaluate the makespan} \]
\[ \text{If } c(n) < c(n-1) \text{ then } LA, \text{Signal}(0) \text{ for all } 0 \leq j < \tau \]
\[ \text{Else } LA, \text{Signal}(1) \text{ for all } 0 \leq j < \tau \]
\[ \text{If } \text{no change in makespan occurs for 150 iterations or 10000 iteration is over } \text{then exit} \]
\[ \text{End} \]

Figure 4. General procedure for learning automata model

4 The Proposed Case-Based Reasoning Model

This section describes the proposed case-based reasoning model, CBR-LA, in details. At beginning, concepts in CBR are briefly reviewed.

In CBR, a set of cases stored in a case base is the primary source of knowledge. Cases represent specific experience in a problem-solving domain, rather than general rules. The main activities when solving problems
with cases are described in the case-based reasoning cycle[19]. This cycle proposes the four steps: retrieve, reuse, revise, and retain. First, the new problem to be solved must be formally described as a case (new case). A case consists of three parts: Problem situation including the initial conditions and the goal, solution and the performance. Then, a case that is similar to the current problem is retrieved from the case base. In the next step, the solution contained in this retrieved case is reused to solve the new problem; i.e., the solution is adapted in order to come to a solution of the current problem. Thereby, a new solution is obtained and presented to the user, who can verify and possibly revise the solution. The revised case (or the experience gained during the case-based problem solving process) is then retained for future problem solving; e.g., the case can be stored in the case base. The last step realizes the learning phase of a CBR application. It should be noted that CBR does not offer a definite solution but proposes hypotheses and ideas to go through the solution space.

There are various approaches for solution adaptation where transformation and combination are among well-known techniques. In the proposed model, the adaptation, which is performed by means of learning automata, is a hybrid combinational transformational adaptation. Figure 5 shows the schematic of the proposed case-based recommender. The three parts of a case are ETC, Mapping and makespan; which is denoted by a 4-tuple (ETC, π, C). To retrieve similar cases, similarity criterion is defined as the similarity between two matrices; ETC of the newly arrived metatask and ETC of the stored cases. The similarity function $H: (ETC, ETC) \rightarrow r \in \mathbb{R}$, returns the Euclidian distance of two matrices; a quantity indicating the amount of similarity between two matrices. In general, calculating Euclidian distance of two matrices of size $|S| \times |M|$ (assuming that they are of the same dimension) has the time complexity of order $O(|S| \times |M|)$. Seeking in a case-base storing $|E|$ cases, the computation cost is $O(|S| \times |M| \times |E|)$, which is computationally expensive. Regarding this fact, we calculate the Euclidian distance of first columns of two matrices as an approximate distance of two matrices. In this way, the computation cost confines to $O(|S| \times |E|)$.

![Figure 5: The proposed case-based recommender model](image)

Similarity between the new case and a stored case is defined as:

$$r = \sqrt{\sum_{i=1}^{3} (ETC \{0\}[i] - ETC \{0\}[i])^2} \quad ETC \in E$$

where $ETC$ is the ETC of new case. Due to the fact that tasks in a metatask are arbitrary, without loss of generality, it is assumed that an $ETC$ matrix is sorted on the first column on a descending bases in order to make the Euclidian distance meaningful as a similarity criterion, i.e $ETC \{0\}[i] \geq ETC \{0\}[j] \quad \forall i < j < \tau$.

Let $E$ be the set of stored cases in the case-base and $\Theta$ be the set of selected cases. A number of cases with smallest $H$ $(ETC, ETC \in E) \forall S_i \in E$ are selected and retrieved. After retrieval of a set of cases, adaptation phase starts.

Adaptation is the process of transforming the map of retrieved cases in order to find a mapping for the newly arrived metatask. This process is done by a learning automata model. Learning automata model described in previous section is redefined as follows. The automaton assigned to task $S_j^{\varphi_j}$ is represented as a 3-tuple, $(\alpha_j^{\varphi_j}, \beta_j^{\varphi_j}, A_j^{\varphi_j})$, where $\varphi_j \in \Theta, 0 \leq j < |\Theta|$. To find a solution, every task $s_j^{\varphi_j}$ belongs to case $\varphi_j$ is associated with a variable structure learning automata $(\alpha_j^{\varphi_j}, \beta_j^{\varphi_j}, A_j^{\varphi_j})$ as explained in the previous section. If
is mapped to a machine, associated LA sets biased to the mapped machine \( m_j \). It is done by initializing the LA probability vector near one for the action corresponding to \( m_j \). For a biased LA, the rate of penalizing is much greater than the rate of rewarding. It helps a wrongly biased automaton correct itself rapidly. On the other hand, if \( s_i^{\phi_j} \) is not mapped to a machine (i.e. in the retrieved case no corresponding machine is defined for it), associated LA is set unbiased, that is

\[
\text{if } \pi^{\phi_j}(s_i^{\phi_j}) \text{ is available then let } \\
\alpha_{\phi_j}^{s_i} = \beta_{\phi_j}^{s_i} \\
A_{\phi_j}^{s_i} \cdot p(q) \approx 1, \text{ for } m_q = \pi^{\phi_j}(s_i) \\
A_{\phi_j}^{s_i} \cdot p(q) \approx 0, \text{ for } m_q \neq \pi^{\phi_j}(s_i) \\
\text{if } \pi^{\phi_j}(s_i^{\phi_j}) \text{ is not available then let } \\
\alpha_{\phi_j}^{s_i} = \alpha' \\
\beta_{\phi_j}^{s_i} = \beta' \\
A_{\phi_j}^{s_i} \cdot p(x) = A_{\phi_j}^{s_i} \cdot p(y), \forall x, y 0 \leq x, y < |M|
\]

Learning automata model starts iterating in the way explained in the previous section until one of the termination conditions is fulfilled. Figure 6 is the general procedure for the proposed case-based recommender model.

```
Begin
  Find n closest cases \( c_i \in E \) to ETC and put them in \( \Theta \)
  For each case \( s_i \in \Theta \)
    Set bias learning automata to the mapping of case \( s_i \)
    Resolve mapping
  Return the case \( s_i \in \Theta \) with minimum makespan
End
```

Figure 6. General procedure for case-based recommender model

The mapping found by learning automata model is used for the next step, proposing a solution. Mappings derived from \( \Theta \) are compared and the best one \( (\rho_{\text{min}} = \min(\ C^{\phi_j}) \ 0 \leq j < |\Theta|) \), one that has the minimum makespan, is selected as the final solution. If the final solution has a major difference with its original case, it is stored as a new case.

Learning automata described in [1] on its own, needs a great effort to be exerted to find a semi-optimum mapping. Our proposed model, on the other hand, due to starting from a point near the solution in the solution space, reduces number of iterations. In the next section, our assertion is investigated through some experiments.

5 Experimental Results

In this section, a set of various metatasks are tested against the proposed case based recommender and the impact of case-based reasoning on the reduction of the number of iterations and the makespan is shown. It begins with an introduction to the simulation environment and subsequently the results are presented and discussed.

5.1 Simulation Environment

For the simulation studies, characteristics of the ETC matrices were varied in an attempt to represent a range of possible HC environments[6]. The ETC matrices used were generated using the following method. Initially, a \( \tau \times 1 \) baseline column vector, \( B \), of floating point values is created. Let \( \omega_h \) be the upper bound of the range of possible values within the baseline vector. The baseline vector is generated by repeatedly selecting a uniform random number, \( x_1 \in [1, \omega_h) \), and letting \( B(i) = x_i \) for \( 0 \leq i < \tau \). Next, the rows of the ETC matrix are constructed. Each element \( ETC(t_i, m_j) \) in row \( i \) of the ETC matrix is created by taking the baseline value, \( B(i) \), and multiplying it by a uniform random number, \( x_i^{\phi_j} \), which has an upper bound of \( \omega_r \). This new random number, \( x_i^{\phi_j} \in [1, \omega_r) \), is called a row multiplier. One row requires \( \mu \) different row multipliers, \( 0 \leq j < \mu \). Each row \( i \) of the ETC matrix can then be described as \( ETC(t_i, m_j) = B(i) \times x_i^{\phi_j} \), for \( 0 \leq j < \mu \). (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 $\omega_i \in [1, \omega_h \times \omega_p] \times [1]$. [6]

To generate different mapping scenarios, the characteristics of the ETC matrix were varied based on several different methods from [5]. 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 $\omega_p = 3000$ and low task heterogeneity used $\omega_p = 100$. Machine heterogeneity represents the variation that is possible 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 machine heterogeneity values were generated using $\omega_i = 1000$, while low machine heterogeneity values used $\omega_i = 10$. 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.

To further vary the characteristics of the ETC matrices in an attempt to capture more aspects of realistic mapping situations, different ETC matrix consistencies were used. An ETC matrix is said to be consistent if whenever a machine $m_j$ executes any task $t_i$ faster than machine $m_k$, then machine $m_j$ executes all tasks faster than machine $m_k$ [6]. Consistent matrices were generated by sorting each row of the ETC matrix independently, with machine $m_0$ always being the fastest and machine $m_{\mu-1}$ the slowest. In contrast, inconsistent matrices characterize the situation where machine $m_j$ may be faster than machine $m_k$ for some tasks and slower for others. These matrices are left in the unordered, random state in which they were generated (i.e., no consistency is enforced). In both matrix types, the condition $ETC[0][i] \geq ETC[0][j]$ $\forall i < j < \mu$ exists.

Eight different cases for ETC matrix characteristics are used in this study: high or low task heterogeneity, high or low machine heterogeneity, and one type of consistency (consistent, inconsistent) which are shown in Table 1. In this study we use ETC matrices with $\tau = 512$ tasks and $\mu = 16$ machines.

5.2 Experiment No.1

In this experiment the impact of combination of case-based reasoning and learning automata over iteration and makespan is tested. The iteration needed to resolve a mapping is compared with the model that uses learning automata only where LA model uses reinforcement algorithm $LR_\alpha$ with $\alpha = 0.1$ and $\beta = 0.01$. As termination condition, learning automata is iterated until no enhancement in makespan is made for 150 iterations, or the number of iterations exceeds a pre-specified maximum limit.

The results tabulated in Table 1., where the adaptation is done by $LR_\alpha$ learning automata with $\alpha = 0.05$ and $\beta = 0.1$, and case-base is filled by 100 cases, and the result of each experiment is the average of 20 runs. As shown in Table 1., in the worst case i.e. experiment No.7, at least 10 percent reduction in the number of iterations is gained and it peaks to 17 percent in experiment No.2. Since, performing each iteration is computationally expensive; even 10 percent reduction in the number of iterations is a significant saving.

<table>
<thead>
<tr>
<th>Experiment No.</th>
<th>Machine Heterogeneity</th>
<th>Task Heterogeneity</th>
<th>Consistent</th>
<th>LA Model</th>
<th>CBR Iteration</th>
<th>Iteration Enhancement</th>
<th>MakeSpan Increase Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
<td>Low</td>
<td>Yes</td>
<td>765.8</td>
<td>668.2</td>
<td>97.6</td>
<td>4.81%</td>
</tr>
<tr>
<td>2</td>
<td>Low</td>
<td>High</td>
<td>Yes</td>
<td>766.1</td>
<td>637.2</td>
<td>128.9</td>
<td>3.82%</td>
</tr>
<tr>
<td>3</td>
<td>High</td>
<td>Low</td>
<td>Yes</td>
<td>760.5</td>
<td>660.5</td>
<td>99.9</td>
<td>3.60%</td>
</tr>
<tr>
<td>4</td>
<td>High</td>
<td>High</td>
<td>Yes</td>
<td>773.6</td>
<td>675.8</td>
<td>97.8</td>
<td>4.19%</td>
</tr>
<tr>
<td>5</td>
<td>Low</td>
<td>Low</td>
<td>No</td>
<td>788.7</td>
<td>689.8</td>
<td>98.9</td>
<td>4.97%</td>
</tr>
<tr>
<td>6</td>
<td>Low</td>
<td>High</td>
<td>No</td>
<td>804.3</td>
<td>710.4</td>
<td>93.9</td>
<td>3.50%</td>
</tr>
<tr>
<td>7</td>
<td>High</td>
<td>Low</td>
<td>No</td>
<td>780.5</td>
<td>699.8</td>
<td>80.8</td>
<td>4.03%</td>
</tr>
<tr>
<td>8</td>
<td>High</td>
<td>High</td>
<td>No</td>
<td>789.7</td>
<td>696.4</td>
<td>93.4</td>
<td>4.01%</td>
</tr>
</tbody>
</table>

Table 1. Iteration enhancement for each model (Case-base of size 100)

Results also state that makespan has increased at most 5 percent in experiment No. 5 but the number of iterations for the same experiment is decreased by 12.5 percent. Despite a slight increase in makespan, a significant decrease in number of iteration is gained, which is the main objective of the proposed model.
5.3 Experiment No.2

As mentioned in the previous section, to adapt a case to the problem, a learning automaton is assigned to each task, and set biased to the mapping recorded in the case. This may raise a question: setting LA biased to an action, can it escape from it? The answer is shown in Table 2. The experiment is made for 20 cases with low task and machine heterogeneous ETC. It is seen that 95 out of 512 assignments changed after adaptation. This shows that wrongly biased LAs can correct themselves.

<table>
<thead>
<tr>
<th>Case No.</th>
<th>Number of Iterations reduced</th>
<th>changed</th>
<th>unchanged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53</td>
<td>96</td>
<td>416</td>
</tr>
<tr>
<td>2</td>
<td>168</td>
<td>92</td>
<td>420</td>
</tr>
<tr>
<td>3</td>
<td>-32</td>
<td>101</td>
<td>411</td>
</tr>
<tr>
<td>4</td>
<td>136</td>
<td>100</td>
<td>412</td>
</tr>
<tr>
<td>5</td>
<td>189</td>
<td>93</td>
<td>419</td>
</tr>
<tr>
<td>6</td>
<td>151</td>
<td>110</td>
<td>402</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>97</td>
<td>415</td>
</tr>
<tr>
<td>8</td>
<td>214</td>
<td>85</td>
<td>427</td>
</tr>
<tr>
<td>9</td>
<td>-23</td>
<td>92</td>
<td>420</td>
</tr>
<tr>
<td>10</td>
<td>26</td>
<td>85</td>
<td>427</td>
</tr>
<tr>
<td>11</td>
<td>-17</td>
<td>95</td>
<td>417</td>
</tr>
<tr>
<td>12</td>
<td>288</td>
<td>94</td>
<td>418</td>
</tr>
<tr>
<td>13</td>
<td>38</td>
<td>105</td>
<td>407</td>
</tr>
<tr>
<td>14</td>
<td>97</td>
<td>92</td>
<td>420</td>
</tr>
<tr>
<td>15</td>
<td>21</td>
<td>92</td>
<td>420</td>
</tr>
<tr>
<td>16</td>
<td>129</td>
<td>95</td>
<td>417</td>
</tr>
<tr>
<td>17</td>
<td>52</td>
<td>83</td>
<td>429</td>
</tr>
<tr>
<td>18</td>
<td>76</td>
<td>105</td>
<td>407</td>
</tr>
<tr>
<td>19</td>
<td>116</td>
<td>95</td>
<td>417</td>
</tr>
<tr>
<td>20</td>
<td>77</td>
<td>83</td>
<td>429</td>
</tr>
</tbody>
</table>

Table 2. Number of assignment changes after adaptation

5.4 Experiment No.3

This experiment shows that by gradually filling the case base with new cases, the system improves its performance over time. The experiment starts with a case base with only one case and continues until the case base contains 100 cases. This experiment is performed for low task and machine heterogeneity and results reported are the average of 20 runs. Figure 7 shows the difference between the number of iterations made for the retrieved case and the number of iterations made for the newly adapted case. A rapid decline in the difference indicates that the proposed model enhances its performance overtime.

![Figure 7. Iteration reduction over time](image)

6 Conclusion

In this paper, a case-based reasoning model for task assignment in heterogeneous computing system based on learning automata was proposed. Previously stored cases speed up the process of finding a semi-optimized mapping from metatask to HC. The main idea is to use mappings of experienced cases to find a solution for newly introduced cases. By conducting some experiments we showed that the proposed model which is a
combination of the case based reasoning and the learning automata model outperforms the model that only uses learning automata. Experiments showed 10% reduction in the number of iterations needed to find the solution which is significantly cost-effective.

References