A Market-Based CPU Resource Allocation System with Strategic Agents

Wurong Zhu   Saneyasu Yamaguchi   Tay Jet Kiat   Hitoshi Aida
Aida Lab, Department of Frontier Informatics
The University of Tokyo
{zhu, sane, tjk, aida}@sail.t.u-tokyo.ac.jp

Abstract

We have proposed and implemented a market-based idle CPU resource allocation system that utilizes idle CPU resources. In this paper we propose two-price setting strategies. The simulation result shows that both of them work well in terms of price stability, market equilibrium, and efficient resource utilizations.

1. Introduction

The advances in computing and networking technology have enabled the utilization of idle computational resources distributed in the network. Many researches have been focused on this area. We proposed and developed a system called the Background Task Space System (BGTS) [1], which provides a platform to use idle computers. However, it relies on the good will of users to provide their idle resources. Besides, it does not imposes any constrains on resource consumption. It is necessary to control access to idle computational resource. Moreover, today, more and more computers are interconnected with each other because of the pervasive use of the Internet forming an enormous computational power; it is an important issue to make resource owners be convinced that they are able to account for the relative costs and benefits of providing idle computational resources.

Since market facilitates resource management in human societies, researches on applying economic idea to computational resource allocation have drawn a lot of attentions recently. Spawn [2] uses auction to allocate CPU time among tasks competing for CPU time in a distributed network. The Popcorn project [3] goes a step further by providing an infrastructure for global computing over the Internet based on auction mechanisms. These precedent works show that there are potential benefits of using economic principles in managing distributed computing systems.

Underlying these environments, we have designed and implemented a market-based idle computational resources (in particularly, CPU time) allocation system [4]. Resource allocation is modeled in a simple economy, in which resource consumption is required to charge for its usage. Prices are introduced to facilitate resource allocations that are set by resource providers.

Since price serves to regulate demand and supply in market-based systems, how to set up price and the price dynamics have great influence on whether it can bring about efficient resource allocations. If the price fluctuates, it cannot enable resource consumers to make correct scheduling decisions, which will lead the market-based approach to be ineffective. Most previous works employ auction mechanism in which the resource is given to who submits the highest bid and the price tends to oscillate and volatile which has been observed in [2]. Moreover, using economics idea in computer science needs to consider the tradeoff of efforts spent operating the market versus the improvement in performance gained from using market-based approaches. An undesirable property of auction is slow price determination. In an auction, all interested consumers need to communicate with a provider (send bids) before any decisions are made, the provider must then inform all bidders of the acceptance or rejection of their offers. In addition, the resource provider, the auctioneer, has the advantage of over resource consumers in auction while in a perfect competitive economy prices are set by demand for resources.

In this paper we propose two price setting strategies for a *demand-driven provider agent* and a *learning-based provider agent* based on interactions between the resource provider and the resource consumer where consumer pays for the resource if it decides that the price is appropriate while provider sets up price according to consumer's response to price. The results of simulations show *Demand-driven provider agent* works well in setting price when resource provider wishes to process multi-tasks while a *Learning-based provider agent* can adaptively set up price when resource provider wishes to process single-task. Both of them are able to lead the market to a steady state near to market equilibrium which enables efficient resource allocations.

The rest of this paper is organized as follows. Section 2 introduces the market-based CPU resource allocation system. We describe the criteria used to evaluate the proposals in Section 3. And then, we present two price-setting strategies in Section 4 and 5 in which simulation results will be shown. Finally, we give concluding remarks and suggested directions for future work.

2. A Market-Based CPU Resource Allocation

2.1 The Concept of Market-Based CPU Resource Allocation

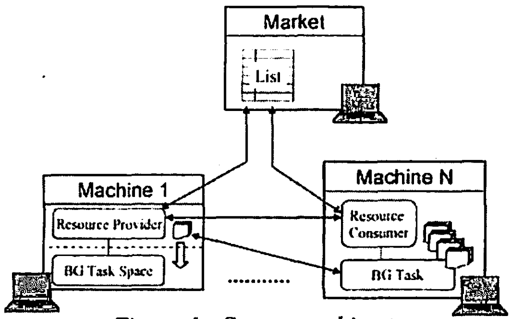

Figure 1: System architecture

The market-based CPU resource allocation system performs distributed resource allocation by employing market-based technique. As figure 1 shows, it consists of three basic entities, *resource providers*, *resource consumers*, and a *market*. A resource provider, an idle computer, advertises its information (price, computer specifications) with the market. A resource consumer, an overloaded computer wishing to rent idle resources gets information from the market attempting to find a suitable resource. Upon decision, it sends the resource provider a request for transactions. After receiving a request for transaction, the resource provider will agree to accept the transaction upon decision. Then, it starts to execute the task, after the completion of execution, resource provider returns result to resource consumer, this results a payment from the resource consumer. The resource provider then updates the corresponding information in the market. If there is no resource consumer requesting for transaction, the resource provider will continually adjust the price and update the corresponding entry in the market so as to be able to make transaction with resource consumers. So in setting prices, resource providers are totally passive, this is exactly the case in an economy with perfect competition. The market provides service to facilitate resource allocation like advertising information and managing account.

Since we focus on investigating using economic idea in resource allocation, we assume that users participating in the system do not have malicious intentions.

2.2 An Economy of CPU Time

2.2.1 The Goods and The Money

Our basic resource allocation unit is a block of CPU time, i.e., the number of instructions used to run a task. For example, a task consumes the same amount of CPU time (number of instructions), even though running it on an under-loaded computer whose processing speed is 2GIPS may take half of the time compared with *running it on an* under-loaded computer with 1GIPS processing power. The CPU resource allocation is carried out in terms of an abstract currency (¥), which represents a form of right with which users can use idle CPU resources.

2.2.2 Resource Provider

We intend to model an idle computer whose user agrees to host tasks from other users in exchange for the currency. Since we are considering the use of idle CPU resources, it is reasonable to make such a simplification that user would rather get his idle computer run tasks for other users to make profit than let it idle on the condition that users are assumed not to take the expenses needed for running computers into consideration. The resource provider sets up price according demand from resource consumers. The algorithm is expressed in provider agent which will be discussed later.

2.2.3 Resource Consumer

To model the user behavior in the BGTS system [1], resource consumer generates a random number of tasks and maintains them in a pool waiting for service. Resource consumer expresses its demand for renting CPU time from resource provider in form of tasks. The task parameters include: the task *complexity*, the amount of CPU time it needs to consume represented by the time needed to run it on a baseline computer of speed 1GIPS, and the task *value* which represents the budget the resource consumer can pay for the resources.

Like provider agent, the actual purchase decision and buying activity are handed over a consumer agent in which user's buying strategy is coded. At the present, we implemented an agent which purchases the lowest priced CPU resources. We use a queue to maintain these tasks; this is a priority queue maintained based on the ratio of task *value* per task *complexity* and resource consumer makes purchase decision by giving high execution priority to tasks with high priorities.

2.2.4 Market Equilibrium

The basic question about the market-based approach is whether it works in terms of economical efficiency, i.e., market equilibrium. The market equilibrium is reached at the intersection of the demand and supply curve where supply equals to supply. To facilitate discussion, we define system load as the expected amount of processing requested per time unit divided by the total amount of processing power in the system. In formula form, it is:

$$L = \lambda / \mu T \tag{1}$$

where $1/\lambda$ is the mean time of task arrival interval, $1/\mu$ is the mean time of individual task complexity, T is the total processing power in the system, and L is the system load. It is obvious that the system load L represents the relative demand to supply. In an overloaded system, only 1/L of total tasks will be processed. The market equilibrium p_{eq} can be calculated as (2), where f(p) represents the probability density function of every task's *value* to *complexity* ratio.

$$\int_{Peq}^{\infty} f(p)dp = \mu T / \lambda \qquad (2)$$

3. Evaluation Metrics

Since the actual resource allocation is carried by resource consumers and resource providers, the way the resource provider sets up prices is quite sensitive to whether the market mechanism works well. We have designed two provider agents; a *demand-driven provider agent* and a *learning-based provider agent*. Our main interest is whether the market mechanism can bring about efficient resource allocation. Thus, we will evaluate our proposal with respect to the following criteria:

1. Market equilibrium, price stability, and price adaptation
2. Resource utilization efficiency

We are interested in whether the price can reach market equilibrium calculated by formula (2). Price stability is crucial to ensure resource consumers' scheduling stability. If the price fluctuates wildly, resource consumers that base their decisions on the state of price will follow suit, leading to poor performance. We are also interested in whether the price responds in a reasonable way to the changes of relative demand and supply. Resource utilization efficiency measures how effective the market-based approach works to allocate resources. If the overhead needed for market operation, i.e., price adjustment and transactions between resource providers and resource consumers, is too high, it means the market approach is not succeeding in efficient resource allocation.

4. A demand-driven provider agent

4.1 Price Setting Strategy

The *demand-driven provider agent* wishes to process multi-task who needs to denote the maximum number of tasks it wishes to take in simultaneously[1]. It raises and lowers the price according to an intuitive algorithm; if the asking price is accepted by a resource consumer, it raises the price; on the other hand, if none of the resource consumers responds to the price, it lowers the price until one of the resource consumers accepts the price and asks for transaction. The price adjustment action is event driven.

The *demand-driven provider agent*'s strategy is parameterized by an *initial asking price, the maximum number of tasks* and an *up/down ratio*, where the up/down ratio indicates an upper and lower limit that a resource provider wishes to adjust each time. The current asking price is simply expressed by multiplying the previous asking price by a price adjustment ratio shown as

$$currentPrice = ratio * previousPrice \qquad (3)$$

Where the price adjustment ratio is expressed in a deterministic function of the number of tasks currently processed by resource provider shown as follows:

$$ratio = downRatio$$
$$+ taskNum * (upRatio - downRatio) / \max TaskNum \quad (4)$$

The ratio increases from the lower to the upper limit linearly in proportion to the number of tasks currently running. If the tasks currently running on the resource provider are more than half of the maximum number of tasks, the agent raises the price upon considering that is still a chance of setting up higher price; otherwise, the agent will lower the price to avoid missing the chance of transactions. The purpose of updating the price adjustment ratio is to resolve two contradictory goals; attempting to make transactions at a high price is desirable while a high price may lead resource consumers not being able to afford for resources.

4.2 Simulations

Simulation Conditions: At first we made a simple setting; 6 resource providers and 1 resource consumer were involved in the market. Each resource provider used a *demand-driven provider agent,* the resource consumer made purchase decisions by selecting the cheapest CPU resource to run the most valuable task as described in Section 2. Each resource provider's maximum number of tasks is 6.

Tasks were generated on the resource consumer following a Poisson distribution whose mean arrival interval was $1/\lambda$, which was fixed at 1000ms. Individual task *complexity* was assumed to follow an Exponential distribution whose mean time was $1/\mu$, and individual task *value per complexity* was drawn from a Gaussian distribution with a mean of 5 ¥/GI and a standard deviation of 1.

The resource providers were identical computers whose processing power was 1.5 times as fast as the baseline computer (1GIPS), hence yielding a total computational power of 9GIPS. Thus according to formula (1), it is easy to compute system load *L*. We have conducted several experiments under different system load which is achieved by adjusting μ. Table 1 gives the experiment conditions; experiment A, E, F, G were carried out under varying system load while

[1] We make a simplification by assuming that the amount of CPU resource consumed by each task to be one Nth of the total available resource, where N represents the number of concurrently running task. A research about precise measurement of the number of instructions executed by each thread can be found in [5].

experiment A ~ D were done with different price adjustment ratio (*up/downRatio)*.

Exp	$1/\mu$ [ms]	$1/\lambda$ [ms]	L	UpRatio /DownRatio
A	20,000	1000	20/9	0.03%
B	20,000	1000	20/9	0.3%
C	20,000	1000	20/9	3%
D	20,000	1000	20/9	10%
E	10,000	1000	10/9	0 03%
F	30,000	1000	30/9	0 03%
G	40,000	1000	40/9	0.03%

Table 1: Experiment parameters

4.3 Results

Price Stability and Market Equilibrium: Experiment results are shown in Table 2. We use the average price (*Price-Avg.)* to compare with the market equilibrium, which is calculated from transaction prices of each resource provider after the price has reached the steady state. To measure the price stability, we calculate the standard deviation of the price (*Price-Stdev*). We also give a column to show the market equilibrium (p_{eq}) calculated according to formula (2), and how much difference the average price from it is shown in *Diff* column. Figure 2 is an example for transaction price convergence to steady state. As it shows, the transaction prices are low during start-up. Prices increase until equilibrium is reached where it fluctuates within a certain band (indicated by *Price-Stdev*). We also note that when the price adjustment ratio increases, the value of *Price-Stdev* becomes large.

Exp	Price_Avg [¥/GI]	Peq [¥/ GI]	Price_Stdev	Diff
A	5.216	5.101	1.66%	2.225%
B	5 518	-	3.65%	8.175%
C	5 647	-	7.56%	10.703%
D	5 658	-	9.30%	10.912%
E	4.310	3.718	1.99%	15 923%
F	5.701	5.500	1.68%	3.654%
G	5.947	5.739	1.25%	3 624%

Table 2: Experiment results

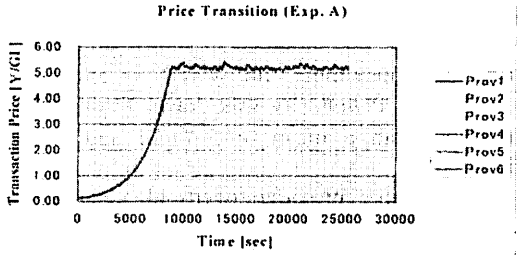

Figure 2: Price transition

Price Response to Changes in the Relative Demand and Supply: According to the results of experiment A, E, F and G; the increase of demand results in an increase in the average price. We can see that the system adapts to changes in demand and the price is meaningful.

Resource Utilization Efficiency: We use the CPU utilization efficiency, i.e., the percentage of CPU time that is used to run tasks, to measure the resource utilization efficiency. In each experiment, a value of about 87% has been achieved; when the maximum number of tasks is increased to 20, the value rises to about 96%. This tells that the pricing mechanism is efficient in terms of resource utilization.

5. A learning-based provider agent

Since there may be different requirements from resource consumers, e.g., a consumer with deadline tasks may wish to buy resources from such a provider who runs single task so that it can use the CPU time exclusively to meet task deadline. Therefore, we need to investigate the strategy for setting up price in processing single task. In this case, if resource provider adjusts the price solely according to the number of tasks currently running which is 1 or 0, it may raise or lower the price extremely which let the price tend to fluctuate. This is not favorable for both resource providers and resource consumers; the resource provider as an idle computer wishes execute as many tasks as possible to make profit while the resource consumer has the needs to get tasks processed as many as possible provided that they can afford.

This inspires us to investigate an adaptive pricing algorithm. The number of tasks currently running shows whether the current asking price is appropriate. Besides, the resource consumer's response to the price adjustment action is also important. For example, if it raises the price at a certain price level and this action tends to lead no transactions from resource consumers, the provider agent should lower the possibility of taking this action at the same situation. The idea is the same as what we do in our daily life; we learn by interacting with our environment. The computational approach to learning from interaction is reinforcement learning that deals with learning what to do – how to map situations to actions – so as to maximize a numerical reward signal.

5.1 Model

We have designed a *learning-based provider agent* by adding the reinforcement learning ability in its price setting strategy. The price adjustment can be modeled as: the provider agent interacts with the environment (the market) through taking *actions* (adjust price), the market responds to the *action*, accept or reject it and presents new situations (*state*) to the agent.

One of the most distinctive features of reinforcement learning is using "*reward*" to formalize the idea of a goal. The provider agent's goal is to achieve proper price setting policy. We use reward to evaluate how good taking action a in state s is for a provider agent. Through interaction with the market, the provider agent learns how to change its price adjustment policy as a result of its experience. Details about the reinforcement learning can be found at [6]. Thus, we need to define "*state*", "*action*" and "*reward*" in our pricing model.

State: To describe a "state", we assume that the resource provider has the pre-knowledge of the maximum and minimum price of the market; therefore we can divide the price range into different price levels.

A "state" is described by a combination of a price level and whether the price is accepted (we call it a "busy" state, in which the provider is running a task) or not (we call it an "idle" state) shown as follows.

$$state = (price\_level,\ busy\ or\ idle) \qquad (5)$$

For example, if the price is known to be greater than 0￥/GI and less than 5￥/GI, we can divide the price range into 5 levels and define 10 states like:

$$"state\ 0(0\sim 1¥\ /\ GI\ ,busy\ )"$$
$$"state\ 1(0\sim 1¥\ /\ GI\ ,idle\ )"$$
$$"state\ 2(1\sim 2¥\ /\ GI\ ,busy\ )"$$
$$\cdots$$
$$"state\ 8(4\sim 5¥\ /\ GI\ ,busy\ )"$$
$$"state\ 9(4\sim 5¥\ /\ GI\ ,idle\ )"$$

Action: The "action" is the current price decision, which is to raise or lower the price and by how much percantage the price will be adjusted. Like the *demand-driven provider agent,* we specify sevaral price adjustment ratios for the *learning-based provider agent* to choose. Thus, the price adjustmen ratio is a direct mapping from the action chosen. A new asking price is calculated according to (3).

Reward: .For a provider agent, a good action is setting up a high price that can result in transactions from resource consumers. If an action, i.e., price adjustment, leads to a transaction, it is praised by giving its asking price as immediate reward; otherwise, it is penalized by giving a zero reward. The reward is defined as:

$$\begin{aligned} &if\ \ (asking\_price\ is\ accepted) \\ &\qquad reward = price \\ &else \\ &\qquad reward = 0 \end{aligned} \qquad (6)$$

Most reinforcement learning algorithms are based on estimating *value functions* - functions of states (or of *state-action* pairs) that estimate the reward that can be expected. We use a *state-action* function known as Q learning algorithm [6] to estimate how good it is to a provider agent for taking action *a* in state s:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha * (r_t + \gamma * \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)) \qquad (7)$$

Where Q(s,a) represents the discounted long-term expected reward s, γ is the discounting parameter and α is the learning rate parameter.

5.2 Price Setting Strategy

With the above defined *state* and *action*, the *Q*(*s*,*a*) function is represented by a lookup Q table containing a value for every possible *state-action* pair. The table entries are initialized to arbitrary values. Then the procedure for price adjustment is as follows:

1. In a certain state, choose an action *a* corresponding to the maximum Q value to set up price with probability $1-\varepsilon$ and a random action is chosen with ε probability. The price is calculated based on the chosen action *a* according to (3).
2. Observe the response from the market to decide the the successor state s_{t+1} according to (5) and give an immediate reward *rt* according to (6) to evaluate action *a*. Update the *Q*(*s*, *a*) of state-action pair (*s,a)* according to (7) where the max operation represents choosing the optimal action a_{t+1} among all possible actions in the successor state s_{t+1}.
3. Return to step 2.

The *learning-based provider agent*'s strategy is parameterized by an initial asking price, definition of state and actions, and related Q learning parameters.

5.3 Simulations and Results

Simulation Conditions: We conducted experiments by involving 4-resource providers with 2.5GIPS processing power and 4-resource consumers in the market. Each resource provider used a *learning-based provider agent.* The way the resource consumer makes purchase decisions is the same as that of Section 4.1.2. Task generation followed the same distributions as that in Section 4.2. The mean time ($1/\mu$) of individual task's *complexity* was set to 100,000 ms, and individual task's *value per complexity* was set to a mean of 3 ￥/GI with a standard deviation of 1. Table 3 gives the experiment conditions; experiments H, I and J were carried out under varying system load.

The state is divided into 5 price levels from 0￥/GI to 5￥/GI by 1￥/GI interval, and there are 5 actions (a0~a4) with different *up/downRatio* of (+5%, +1%, 0%, -1%, -5%). α, γ and ε were all set to 0.1, the Q table is initialized to 3.

Simulation Results: Table 3 also shows the simulation results. We can see that the *Price-Avg* largely complies with market equilibrium under different system loads. The price also shows adaptations to changes of demand; the price rises when the relative demand increases.

Exp.	$1/\mu$ [ms]	$1/\lambda$ [ms]	L	Price_Avg. [¥/GI]	Peq [¥/ GI]	Price_Stdev	CPU_ UtilRatio
H	100,000	20,000	2	3.14	2.988	5.600%	98.120%
I	100,000	10,000	4	3.653	3.660	9.773%	98.162%
J	100,000	5,000	8	4.138	4.126	3.697%	98.365%

Table 3: Experiment Parameters and Results

The significant result is that resource utilization efficiency (indicated in the CPU_UtilRatio column) is extremely high; this results from learning algorithm. Since provider agent's price adjustment action is evaluated in form of *reward*, which is defined as rewarding *busy* state while penalizing *idle* state. A failure action may lead its Q value to become small; hence the provider agent will avoid taking this action. The Spawn [2] system using auction mechanism to allocate resources in distributed network operates with 7.6% overhead[2].

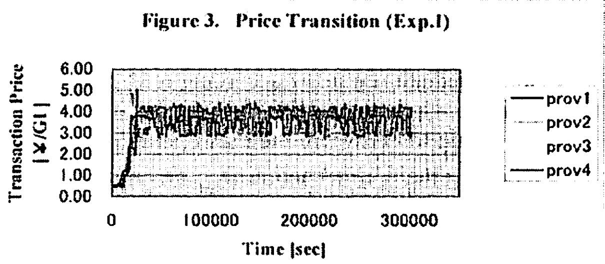

Figure 3: Price transition

5.4 Refinement of State

Figure 3 shows the transaction price of Exp. I, the price tends to fluctuate which suggests us to divide the state by a finer interval. We then refine the state with an interval of 0.2 ¥/GI. From Figure 3, we assume the market equilibrium is between price level 3 ¥/GI and 4 ¥/GI, thus the state is defined as:

"*state* 0(0～3¥ / *GI* , *busy*)"	"*state* 1(0～3¥ / *GI* , *idle*)"
"*state* 2(3.0～3.2¥ / *GI* , *busy*)"	"*state* 3(3.0～3.2¥ / *GI* , *idle*)"
"*state* 4(3.4～3.6¥ / *GI* , *busy*)"	"*state* 5(3.4～3.6¥ / *GI* , *idle*)"
"*state* 6(3.6～3.8¥ / *GI* , *busy*)"	"*state* 7(3.6～3.8¥ / *GI* , *idle*)"
"*state* 8(3.8～4.0¥ / *GI* , *busy*)"	"*state* 9(3.8～4.0¥ / *GI* , *idle*)"
"*state* 10(4.0¥ / *GI* ～, *busy*)"	"*state* 11(4.0¥ / *GI*～, *idle*)"

The other parameters remains the same, the result is very satisfied, the price tends to approach the market equilibrium with a *Price- Avg* of 3.773 ¥/GI and a *Price-Stdev* of 3.532%. Figure 4 shows the average Q values of one of the provider agents from *state 6* to *state12*. Action index is on X-axis, state index is on Y-axis and Q value is on Z-axis. We can see that the provider agent takes actions of raising price (a0) in *state 6*. When the price is near to the market equilibrium (*state 8, 10*), it alternatively chooses actions a1, a2 and a3, which lead to a stable price. When the price rises to a high level (*state 12*), action 4 is chosen. The agent obtains the policy to set up price with respect to price levels.

For a single agent, the ordinary Q-learning is guaranteed to find the optimal policy. However, in the presence of a population of agents, the problem becomes non-stationary and history dependent, it is not known whether any global convergence can be obtained. Despite the lack of theoretical guarantees, we find that each provider agent has obtained self-consistent policy which enables it to reach market equilibrium.

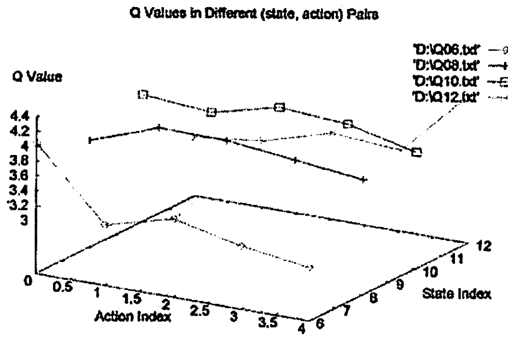

Figure 4: Q values in different (state, action) pairs

6 Conclusions and Future Work

In this paper, we present two price-setting strategies used in our proposed market-based CPU resource allocation system. The results show through setting up price according to consumer's response, the market equilibrium can be reached, and the price is relative stable, particularly the addition of learning ability performs well. Moreover, two strategies show the price determination is quick and the overhead of market operation is low. With elaborated price setting strategies, the market-based approach can display its potential advantage in computational resource allocations. For future work, the obvious extension is to make the *learning-based provider agent* available to deal with setting up price in the case of processing multi-tasks, which may produce further improved results; in addition, an elaboration of the definition of reward and state is needed.

References

[1] S.Yamaguchi, M.Takimoto, H.Aida, and T.Saito. Cooperative background task spaces and its evaluation. In The 4th International Workshop on Network-Based Information System (NBIS '2001), September 2001.

[2] C.A.Waldspurger, T.Hogg, B.A.Huberman, J.O.Kephart, and S.Stornetta. Spawn: A distributed computational economy. In IEEE Transactions on Software Engineering18,2, pages 103 - 177, February 1992.

[3] N.Nisan, S.London, O.Regev, and N.Camiel. Globally distributed computation over the Internet - the popcorn project. In International Conference on Distributed Computing System (ICDS '98), 1998.

[4] S.Yamaguchi, W.Zhu, and H.Aida. A market-based CPU resource allocation system. In the 9th DPS Workshop IPSJ, October 2001. (in Japanese).

[5] O.Hayamizui, K.Taura, and A.Yonezawa. Java bytecode transformation for fine grain CPU resource management. In IPSJ, July 2001. (in Japanese).

[6] R.S.Sutton, and A.G.Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 1998.

[2] It runs auction to receive bids from tasks at a regular time slice of 120,000ms. Each time it executes a single task.