Reliability-Aware: Task Scheduling in Cloud Computing Using Multi-Agent Reinforcement Learning Algorithm and Neural Fitted Q

Husamelddin Balla, Chen Sheng, and Jing Weipeng
College of Information and Computer Engineering, Northeast Forestry University, China

Abstract: *Cloud computing becomes the basic alternative platform for the most users application in the recent years. The complexity increasing in cloud environment due to the continuous development of resources and applications needs a concentrated integrated fault tolerance approach to provide the quality of service. Focusing on reliability enhancement in an environment with dynamic changes such as cloud environment, we developed a multi-agent scheduler using Reinforcement Learning (RL) algorithm and Neural Fitted Q (NFQ) to effectively schedule the user requests. Our approach considers the queue buffer size for each resource by implementing the queue theory to design a queue model in a way that each scheduler agent has its own queue which receives the user requests from the global queue. A central learning agent responsible of learning the output of the scheduler agents and direct those scheduler agents through the feedback claimed from the previous step. The dynamicity problem in cloud environment is managed in our system by employing neural network which supports the reinforcement learning algorithm through a specified function. The numerical result demonstrated an efficiency of our proposed approach and enhanced the reliability.*

Keywords: *Reinforcement learning, multi-agent scheduler, neural fitted Q, reliability, cloud computing, queuing theory.*

Received April 5, 2018; accepted January 28, 2020
https://doi.org/10.34028/iajit/18/1/5

1. Introduction

Large amount of data and systems in cloud environment are driven by the internet and provide massive information which it can be useful for developing applications and dictions making. Resources and applications in the cloud environment are rather heterogeneous and in a dynamic status, these heterogeneity and dynamicity management needs special care to deal with [20]. Since there are huge resources in a dynamic changes environment such as cloud environment, multi-agent resources allocation is essential and it is one of the most significant issues in cloud computing [6]. The resources in cloud computing have been developed and increased in the recent decades which emerges the decentralization management for these resources is essential for the reliability enhancement. There are many capabilities provided by cloud computing such as sharing various king of heterogeneous resources which are distributed among different locations on the cloud environment. A coordinated job scheduling plays an important role in cloud computing where the dynamic changes in the resources takes place [31]. The efficient algorithm for task scheduling can enhance and boost up the performance of system if there is a c suitable policy used for the scheduling. Load balancing plays an important role in cloud computing which can be defined as complementing the all assigned jobs in appropriate manner and time. An effective scheduling method such as adaptive scheduling for heterogeneous resource scheduling is required to manage the cloud resources efficiently.

There are many advances have been done in the cloud computing in the recent decade. Man reinforcement learning-based scheduling approaches have been developed for enhancing the performance in the cloud environment, the scheduling approaches either using a model-free or a mode-based approach. In addition, there are many task scheduling algorithms have been studied in the distributed system environment [5]. On the other hand, the demand of reliability enhancement on distributed and cloud system becomes more and more as the increases of dynamic changes in a complex and massive networked environment such as cloud computing. Improving the resources scheduling in the network affects positively the reliability and decisions taking [23]. The taken decisions are not only for resources processing capacity matching of users requests or scheduling, but they also need to consider the behavior and performance of those resources. In the large distributed systems such as cloud systems, services should continue working even in the presence of unpredictable faults and the user requests should not affect the systems performance [20, 28]. The scheduling decisions should automatically tolerate the failure in order to be reliable, besides, providing high availability

On the other hand, centralized schedulers can perform efficiently if there is obtainable global information. However, lack of the scalability can be one of the centralized scheduling problems as well as the lack of fault tolerance capability [17]. To overcome this problem, several decentralized scheduling approaches have been developed but most of them perform as individual policies of scheduling which raises many synchronization problems [7, 33]. The extensive dependency between the resources and schedulers may lead to over communicate and network resource consumption problems. Therefore, managing scheduling with reasonable communication cost in decentralized environment is an essential issue. Taking in consideration this problem, a research has been conducted in [12] where a decentralized dynamic scheduling approach was proposed entitled the Community-Aware Scheduling Algorithm (CASA).

To achieve the adaptation of resources scheduling in an environment containing heterogeneous resources, variation performance, and application diversity employing an adaptive scheduling approach is essential [16]. The aim of Reinforcement Learning (RL) is to provide a mechanism for solving uncertain decision making problems through interacting with the environment which can use data driven method to obtain a near optimal policies [24]. Reinforcement Learning is a model-free approach which can solve many scheduling problems in a dynamic endowment such as cloud computing environment. There are two types of learning approaches for resource scheduling in Reinforcement Learning. One of those two types is based on the gradient learning policy [1, 32] and the other type uses value function based learning [27].

In this work, we propose a Task Scheduling Implementing Multi-Agent NFQ-based Reinforcement Learning (TSMRL) approach for distributed resource scheduling in cloud environment to solve the problem of over communication and single point failure problem. In our proposed approach, the reliability enhanced by employing a multi-agent method for performing the task scheduling and learning the scheduled task efficiency by providing rewards to the agent after each scheduling action taken by the agent. The multi-scheduler in our proposed approach prevents the occurrence of whole system failure and solves the single point failure problem because if a fault occurred in one scheduler, the scheduling system will continue working through the other schedulers. The aspect that simplifies the decision making mechanism in our approach is there is only one learning table used for resources estimation.

2. Problem Definition

The cloud environment is a dynamic environment which can be affected by changes caused by the user requests and demanded resources state. In addition, the heterogeneity in cloud resources and diversity in the applications creates some difficulties in those resources management. Moreover, traditional approaches for resource scheduling in cloud computing cannot solve those problem in a satisfaction level because of the addition overhead that could come from over communication and network resources consumption. Therefore, to overcome those problems, an adaptive scheduling method is required in order to achieve a smooth and reliable task execution in an environment with dynamicity and heterogeneity.

3. Related Work

In the recent decade several researches have been conducted implementing Reinforcement Learning algorithm to solve cloud computing problems. Tesauro *et al*. [26] proposed a hybrid approach combining reinforcement learning algorithm and queuing theory in an open and closed traffic loop. Their proposed approach employed the queue theory for controlling the system while keeping the offline data for reinforcement learning to train on. In their system they used non-linear function for approximation instead of using lookup tables. Farahnakian *et al*. [8] used reinforcement learning for consolidation the virtual machines in cloud environment. The aim of their approach (RL based dynamic approach consolidation) is to minimize the active energy consumption. Bahrpeyma *et al*. [3] developed a model benefits from the prediction of demand and workload in term of discreteness and cumulative load. Their method also uses a knowledge-based engine employing the Ink Drop Spread which implemented in [2] to design an adaptive controller based on reinforcement learning. Guo *et al*. [11] used reinforcement learning to minimize the response time of multi-tier applications by using parameters such as trial-and-error. Their approach uses neural network to control fuzzy working which provides a fast method to reconstruct the neural system using reinforcement learning. Vengerov [27] proposed a general framework in to provide a dynamic allocation for the cloud resource. Their model uses reinforcement learning algorithm integrated with fuzzy rules to allocate in demand resources. They developed their model to be working without considering the existence of a resource allocation policy in the environment. In Zhang *et al*. [32] prosed a multi-agent method to optimize the online resources allocation. The methodology of taking the decision to allocate the resources in their method depends on two connected problems. The first problem is about taking the right decision for allocating the task locally. The second problem in their method is about task routing and how to take a decision for where to forward the task. In their model they employed heuristic strategies to speed up the learning approach and avoid the unwanted weak policies. In Hussin *et al*. [13] prosed a reliability aware model to improve the reliability in cloud environment. They implanted reinforcement learning along with neural network to develop a task scheduler which considers the dynamicity in the cloud environment. Their system benefits from their previous neural network system [9] which they proposed for task scheduling in the grid network. To develop the task scheduling in their grid scheduling they utilizes a fuzzy rules which employ swarm intelligence for knowledge acquisition. Bu *et al*. [4], the authors developed an approach for define an exploration and exploitation method shows the capabilities of agents. The agents collaborate with the environment to increase the learning exploration, and they estimate the state using the applications' information. In the meantime, the utility or Q table (which is updated by the agent) is ordinal and iteratively shared in the state to view and evaluate the efficiency of resources. However, the authors did not cover decisions regarding the dynamic environment of distributed resources. In our proposed system, we adopted RL, which deploys a highly dynamic environment in a distributed structure. Our proposed system can adapt with the changes in the dynamic environment and take the decision regarding these changes. To measure the indicators of performance, such as completion time and waiting time, the Liu *et al*. [19] deployed a Markov request queue model, which was achieved by considering the shared resources among Virtual Machines (VMs) and several failure types. Using RL as a framework, Wu *et al*. [30] proposed a distributed learning algorithm named as Ordinal Sharing Learning (OSL). In their algorithm, each scheduler has a utility table, and this table must be updated in two steps. First, their algorithm uses local rewards to update the table. Second, their approach to update the table, the scheduler uses the utility table of its adjacent [21]. The scheduler drives its table to the neighbor scheduler after updating the utility table in cooperation with other schedulers, and the neighbouring agent completes the update and delivery of the utility table. Vengerov [27] Proposed a general framework and employed RL to perform dynamic resource allocation between multiple entities. This method uses RL combined with fuzzy rule bases and is flexible in that it can be employed in an environment with and without existing resource allocation policies. Compared to aformentioned methods, our appraoches uses tequniques to reduce the communication and network consuption such as a single Q-table for the learning agent. In our method we use a referee scheduler that can conroll the task scheduling thruogh the gained feedbak from all other schedulers in a central Q-table. We also consider the single point failure in our system by allowing all the agents to benifit from the central utility table (Q-table) and send and get the reward from this table. Therefore, our proposed system can takle the single-point failure and reduce the communication cost at the same time.

4. Background and Motivation

In this section we provide a motivation for using RL algorithm and some preliminaries for the implemented techniques in this work.

4.1. Reinforcement Learning

The main idea behind the RL algorithm is learning through the interaction with the environment [24]. There are three main components of RL algorithm which are the state, and the reward. The state produced from the environment where RL is working in, while the agent can be represented by the brain of the algorithm and the decision maker which takes its decision based on the received response from the environment. The response produced from the previous taken action and called the reward which express the value of (state, action) pair [24]. The agent in reinforcement learning technique trying to maximize the reward in every step [15].

The adaptive control approach usually requires a system model. However, the explicit model for resources in the cloud environment is hard to afford because of many factors and domain problems. Therefore, the proposed adaptive control system requires a model-free mechanism such as Q-learning [18]. In the Q-learning method, the system does not require a model for the environment in the decision making process. Moreover, the only required object for agents to take the decision is the current pair of (state, action). Q-learning is one of RL methods that can be employed when there is no condition of domain knowledge need to be provided for the agents initially [29]. Figure 1 shows the basic structure of RL. As shown in the figure, the agent in RL approach learns through the interaction with the environment. The agent perform the action at when it is in the state s_t then receive a reward (response) r_t for the pair (s_t, a_t) and repeat with incremental value r_{t+1} and s_{t+1} in a continue snow balling way.

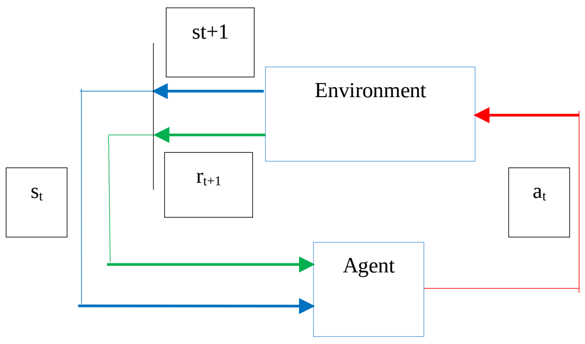


Figure 1. The reinforcement learning basic architecture.

The goal of the learning process in Q-learning is to achieve an optimal policy that can be reflected on the expected cumulative reward named Q-value then followed by taking an action in the current state. This Q-value can be calculated as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (1)$$

Where r_{t+1} is the immediate reward delivered to the agent by taken action a_t from the state s_t, α is the learning rate and γ is the discount factor.

Equation (1) shows the Temporal Difference (TD) approach [25] which is proven to convergence to approximate the function of action-value to an acceptable value taken in account the probability 1 for the target policy. Algorithm 1 shows Q-learning working mechanism by observing the state, obtaining the feedback and update the cumulative reward.

Algorithm 1: Q-learning policy

```
1. Initialize the Q-value table for the pairs (s, a)
2. Repeat for each episode
3. Repeat for each step
4. Observe the current state s
5. Select an action using an exploratory function
6. Obtain action a then get a feedback reward r to reach new
   state s'
7. Update cumulative reward (the Q-value) using equation (1)
8. s = s'
9. End
```

4.2. Reinforcement Learning

The Q-learning rule in the previuos section can be directly implemented in a neural network. In this impelentation (Q-learning with neural network) an error function plays an important rule which aims to measure the error by caculating the difference between the curren Q-value and the previouse one [22]. The approximation function in the continuous domains is essential to estimate the observed experiences in the unseen cases because of the state domain infiniteness. Therefore, artificial neural network is regularly employed to provide support and knowledge base for RL [14]. This kind of approaches that uses the artificial neural network to provide the knowledge support for RL is called as Neural Fitted Q (NFQ) [10].

5. RL-based Task scheduling

The main reasons behind implementing Reinforcement Learning with Neural Fitted Q (NFQ-based RL) in our proposed approach can be defined as follows:

1. The model-based approaches in the scheduling mechanism (which we avoided in our system) depend on the grid information system which delays the resources information acquisition.
2. The dynamicity of the cloud environment produces some difficulties in the cloud resources management, because of that an adaptive method (such as RL) is required.
3. To minimize network communication overhead and reduce the network consumption using a central coordination by NFQ-based reinforcement learning.
4. Q-learning is a flexible and practical for uncertainty and generalization problem solution.
5. We implemented Q-learning because Q-learning is a value iteration method not policy iteration. Policy iteration methods need to train the agent in every episode taking the same policies which not applicable in a dynamic environment such as cloud environment.

In our approach, all of the schedulers can obtain an accurate view of the resources state. To overcome the single point failure problem in our method, all schedulers can undertake the task which allow learning the task by other scheduler if the scheduler in charge failed to schedule the task. The main idea of our proposed approach is to learn the plant function by interaction between the agents and the environment. We need to set a goal to be achieved by the adopted learning mechanism. The goal of the implemented learning method is to achieve better results in task scheduling and resources management (maximize the accepted Q-values) which benefits from the used knowledge based approach. The knowledge based approach exquisite the information from the past experiences to establish a suitable decision. Therefore, more experiences can enrich the knowledge and enhance the decision making. The metrics of our system performance are the successful jobs execution number, response time for jobs execution and resources utilization which are somewhere relevant to our goal. A general structure of our proposed approach is shown in Figure 2.

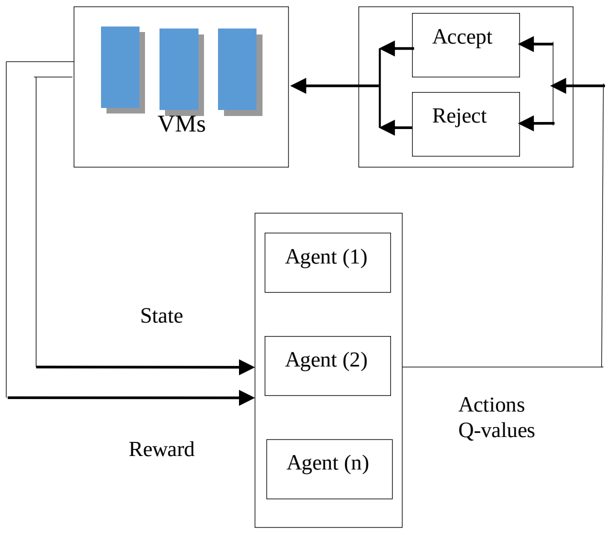


Figure 2. The basic structure of the proposed multi-agent approach.

5.1. Learning Method

The learning method in our appraoch is based on using multiple independing learning agents to accelerate the learning method. There is no need for explicit communications between agents in the independant learning, ulternatively, based on the local reard and state each agent learns independantly. However, this may rise an anomalous resources allocation problem due to the lack of communication between the agents.

To overcome this problem, we implemented nueral ntwork mechanism to work together in a coperative way with reinforcement learning as NFQ-based reinforcement learning. The idea of employing NFQ-based reinforcement learning is to learn the local rewards that submitted by the schedulers agent after collecting those rewards and update the global utility table by the central learner agent.

The learner in our method is responsible of information acquiring and the relation between the pair (state, action). We can classify the learning procosess in our system as a supurvised learning to formulate the model of the system response based on (state, action) thruogh the interaction with the environment. The formulation of task scheduling problem in our system as a supervised learning has a closed relation with the concept of reinforcement learning in which the supervised learning estimate the parameters in adaptive way. We implemented Q-learning for the reinforcement learning process in our system which based on dedicate the reward value to make the decision acording to the pair (state, action). The learnning system recieves a reward in each step and after every action it take, that how the learner learns how to choose the right action to make better results.

5.2. The Environment and State

In our proposed appraoch, the environment under scope is a dynamic envronment (cloud envronment) which rquire special care to deal with. In our system, the state space can be represented as the total number of the nodes available for serving the assigned tasks. Since queuing modeling is one of the important issues in task scheduling, we consider the buffer size queue for each node in the environment. In the dynamic and large system such as cloud systems, it is hard to obtain an accurate information about the resources due to the contiues changes in the resources attributes. Those changes can be rabidly occured and no chance to be adapted by the scheduler and results to incomplete the task within its deadline.

To vercome this problem we impelemneted a queuing theory to control the queue buffer size and the load for each node as a sub queue for each computing serevr or virtual machine and a global queue for users input and output. In our method, we consider the load-aware of nodes represent as rectional to support the knowledge base for the scheduler and accelerate the processing speed. The load of nodes can defined by a function *f*: $Qu \rightarrow L_i$ $qu \in Qu$, the abstract $f(qu) \in L_i$. The definition of this function can be represented as:

$$L = \begin{cases} Qu_t & if\ qu_i = M \\ Qu_s & if\ qu_i \in= (\frac{2M}{3} + 1, M - 1) \\ Qu_m & if\ qu_i \in= (\frac{M}{3} + 1, \frac{2M}{3} \\ Qu_o & if\ qu_i \in (1, \frac{M}{3}) \\ Qu_f & if\ qu_i = 0 \end{cases} \quad (2)$$

Where *M* is the full lenght of buffer size, the remainder buffer size memorey is trivial for Qu_t, semi-trivial for Qu_s, middile for Qu_m, over-middile for Qu_o, and full for Qu_f. The periority of requests assigning will be given to the user requests with full remaining queue buffer size, then over-middile and so on. The total load for each node is a compination of the the remainder queue buffer size in addition to toatal usage of CPU, Bandwidth, RAM and storage in the ith node at a designated moment of time. Based on those node's attributes and jobs expexcted execution time, the scheduler with the aid of NFQ-based reinforcement learning schedule the user requests to the nominated nodes.

5.3. Local Reward

The local rewards in our system are used to reflect the efficiency of job scheduling and the running state. There are differences in job scheduling as the heterogeneity in the virtual machines or nodes. We can represent the jobs need to be scheduled as $Job=[job_1, job_2 , \dots , job_n]$ and the nominated nodes as $VM=[vm , vm_2 , \dots , vm_m]$, where *n* is the number of jobs and *m* is the number of virtual machines. As mentioned, the job can be successfully scheduled only if it meets the specified constrains which are the expected execution time for each job is less than the deadline and load of each node or virtual machine attributes is suitable and it illustrated by following function:

$$f(J_i, VM_j) = \begin{cases} 1 & if\ C(i,j) \le VM_{load} + E(i,j) \\ 0 & if\ C(i,j) > VM_{load} + E(i,j) \end{cases} \quad (3)$$

Where *C(i, j)* is the completion time for each job J_i in a virtual machine VM_j, and VM_{load} is the load for each virtual machine which can be represented by the queue remainder buffer size and the total usage of CPU, RAM, Storage and Bandwidth for each *VM*, *E(i, j)* is the expected execution time for each job J_i in a virtual machine VM_j .

Using markov decision process for the job scheduling, the system's current state at the scheduling moment is simply relevant to the previous state. The action set is represented as 'reject' or 'receive' the job by the virtual machine. As mentioned above, 'receive' is represented by 1 and it is true only if the constraints are met using Equation (4); otherwise, 'reject' and it is represented by 0. The local reward is employed for each scheduler to return the system's running state and the running efficiency of the job scheduling scheme. The local reward in our approach can be represented as the above mentioned function $f(J_i, VM_j)$ as:

$$r(Job_i, VM_j) = \begin{cases} 1\ if\ C(Job_i, VM_j) \le VM_{load} + E(Job_i, VM_j) \\ 0\ if\ C(Job_i, VM_j) > VM_{load} + E(Job_i, VM_j) \end{cases} \quad (4)$$

The job expected execution time can be calculated as:

$$E\left(Job_i,\ VM_j\right)=\frac{\text{job ininstructions}}{\text{VM processing time}}+\frac{\text{job size}}{\text{VM bandwidth}} \quad (5)$$

The completion time for each job J_i in a virtual machine VM_j can be calculated as:

$$C(Job_i,VM_j)=E(Job_i,VM_j)+VM_i availability \quad (6)$$

In each step a scheduler receives number of user requests or jobs for scheduling. A record for each job submission is created by the neural network (more details in next section) for the entry of each submitted job and saved in a scheduled job list. The job submission record contains the information extracted from Equation (5) by the neural network. The agent searches the scheduled job list and acquires the relevant information for each successfully completed job then performs a positive reward for the related resource.

5.4. Neural Network and Action Performing

As we mentioned, one of our system goals is control scheduling tasks in an environment with dynamic changes such as cloud environment. Thus, the control action can be represented as:

$$a=(\Delta S,V) \quad (7)$$

Where *a* is the action control, ΔS is the resources dynamic changes and *V* is the virtual machines placement vision and the resources fitness for the current job scheduling.

The fitness of the resource fitness can be donated by using the L_i extracted from Equation (3) and the resource total usage Resource Total Usage (RTU) which is the utilization of Centeral Processing Unint (CPU), Read Acces Memory (RAM), Storage and Bandwidth at a designated time. The vision of situation to schedule the requests is a combination of resource weight which can be driven from the current VM status and the scheduling policy. Figure 3 shows the implementation of the neural network in our system as NFQ-based reinforcement learning.

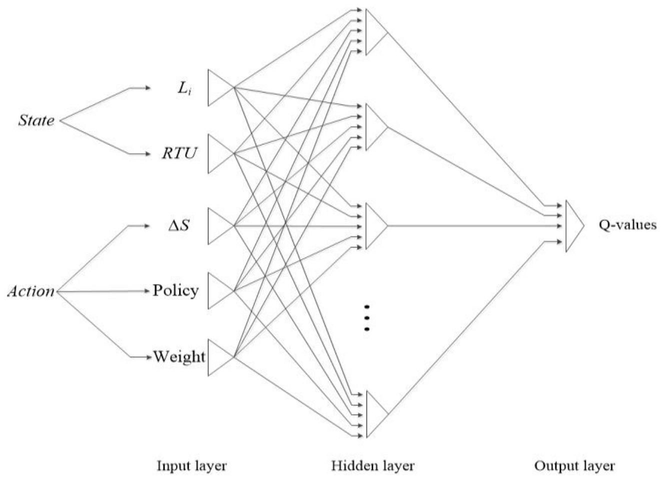


Figure 3. The neural network implementation in the proposed system as NFQ-based reinforcement learning.

Each scheduling agent interact with the environment by obtaining the pair (state, action) at designated moment of time (s_t, a_t). The scheduler agent uses the above neural network (Figure 3) to acquire the (state, action) pairs. The state elements are VM queue remainder buffer size (L_i) and the RTU while the action elements are the resources dynamic changes ΔS, weight and estimated sub-optimal policy. The scheduler agent obtains a sub-optimal policy by employing the Q-learning algorithm and updates the utility table. The proposed system deploys a task scheduling method created by a global task queue with a limited buffer-size and NFQ-based reinforcement-learning task schedulers. In our proposed model, the arrival of the user's requests at the system is modelled as a Poisson process with a mean arrival rate of λ. In a cloud computing environment, one or more servers VMs may be available with mean service rate of the requests μ.

In order to achieve perfect results, the queuing system in our model is structured into three sub-models. The proposed queue model is designed using [9] as a reference.

However, we redesigned the model to have sub-models that interact with each other so that the output of one of them is the input of the other at the same time (Figure 4).

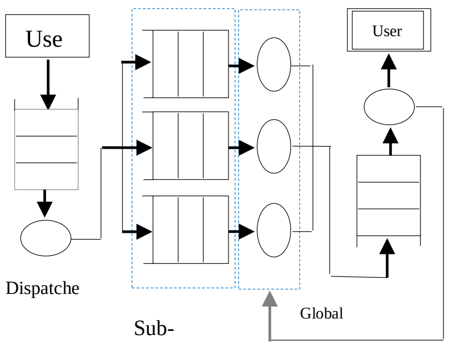


Figure 4. Queuing structure in our proposed model.

The first sub-model of our queue system is represented by the global queue (global receiving) with a task dispatcher. This part of queue is responsible of receiving user's requests and arranging them in a finite queue using Equation (2). As mentioned above, the task arrival is modelled as a Poisson process with a mean arrival λ^{glob} for the first global queue (global receiving). The global queue in our system is modelled as M/M/1 queue model. The mean response time, MR^{glob}, for the first global queue (global receiving) in our model is calculated taking in account the condition $\lambda^{glob} < \mu^{glob}$ as follows:

$$MR^{glob}=\frac{1/\mu^{glob}}{1\text{-}\lambda^{glob}/\mu^{glob}}\Big|_{\lambda^{glob}<\mu^{glob}} \quad (8)$$

The second sub-model (connection) is a set of parallel connections combined with a set of scheduler agents, and each scheduler agent has a sub-buffer queue and a

VM. The scenario of each scheduler agent is as follows: the dispatcher in the first sub-queue dispatches the user's request to the nominated buffer queue. After that, the scheduler picks the user's request from the buffer queue and delivers it to the computing server. Finally, the VM delivers the results after execution to the third part of the queue system (global transition) and then to the central utility Q-table. The second sub-queue of our system (connection) is modelled as M/M/1/m with a buffer size *m*. The probability of assigning a user's request to the *i*th buffer queue is represented by p_i. The mean response time for the user's request to be executed in the *i*th buffer queue is calculated as follows:

$$MR^{conn} = \frac{1/\mu^{glob}}{1\text{-}\lambda_i^{glob} / \mu^{glob}} \Big|_{\lambda^{conn}=p_i\lambda^{glob} \; and \sum_{i=1}^{N} p_i = 1} \tag{9}$$

The conditions $\lambda^{conn} = p_i \lambda^{glob}$ and $\sum_{i=1}^{N} p_i = 1$ are used to ensure the stability of the queue. The stability response time for the user's request to be allocated to any of the buffer queue is given as follows:

$$MR^{conn} = \sum_{i=1}^{n} p_i MR_i^{conn} = \sum_{i=1}^{n} p_i \frac{1/\mu^{glob}}{1\text{-}\lambda_i^{glob} / \mu^{glob}} \tag{10}$$

The third part of our queue system (global transition) is responsible for receiving the executed results from the second sub-queue (connection) and transmit those results to the central learning scheduler as a feedback and then to the cloud user. Similar to the first part of our queue system (global receiving), the third part (global transition) can be also represented as a global queue but with a task transmitter instead of a task dispatcher. Thus, all the tasks involved a global queue twice, once at the arrival and once after the execution. The transmitter receipts the outcome results and transfers them back to the users. The mean of arrival rate at the transition queue (global transition) and at the receiving queue (global receiving) is equal if there are no dropped tasks. The mean service rate for the transmitter is represented by the μ^{trans} conditioning that $\lambda^{glob} < \mu^{glob}$. This part of the queue system (global transition) is modelled as M/M/1, and the mean response time is denoted as:

$$MR^{trans} = \frac{1/\mu^{trans}}{1\text{-}\lambda^{trans} / \mu^{trans}} \tag{11}$$

The total response time in our cloud environment is calculated as follows:

$$MR^{total} = MR^{glob} + MR^{conn} + MR^{trans} \tag{12}$$

As a general scenario for task scheduling, the task dispatcher transfers the user's requests from the main or global queue to the sub-queue. The task dispatching at the designated moment of time executed following specific steps. First, the scheduler agent creates a scheduling policy based on the following parameters:

- The execution of pre-task conditions in VMs.
- The allocation remainder capacity Li of every VM.
- The resource total usage RTU.
- The resources dynamic changes ΔS and weight.
- The prediction of the execution time of the present task.

Second, the *i*th user request is assigned to the nominated sub-queue. Third, each finished and successfully scheduled job produces a positive reward represented by value 1, alternatively each unfinished job produces a negative value represented by 0. Fourth, if there are more than one job submitted by an scheduling agent to the same VM, then the sum of the positive and negative rewards will be calculated and produce a single reward. Finally, each scheduler agent delver its local reward to the central learning agent through the global queue and add it to the cumulative reward then all stored as a knowledge base for NFQ-based Reinforcement learning algorithm.

NFQ-based Reinforcement learning algorithm is responsible for executing the aforementioned steps. According to the state of resources, each scheduler agent takes an action after receiving the reward from the previous action.

5.5. Decision Making

The knowledge base provides information to the agent as we mentioned, this information aid the agent in the decision making processing which is selecting a proper action in each step. The learning agent take an action based on the exploitation (aquired) information, however, the learner also need to execute explorative policies to enhance the performance. There is a tradeoff beteween the couple exploration and exploitation in the propsed NFQ-based Reinforcement learning algorithm because without doing so, the agent will continue in unlimited exploration. In decision making process, the gent searches for the most desirable action which has the most weighted value and high cumulative reward. The action weight can be determined by the status of task which depends on several factors such as the virtual machine queue remainder buffer size, resource total usage and the cumulative reward.

There are three components which are considered in each taken action (dynamicity or resources status change, policy and weight) as shown in Figure 3. In our proposed system, the weight of nominated resource can be affected by the queue remainder buffer size for each VM, expected job execution time and the resource total usage (CPU, RAM, Bandwidth, and Storage). The suboptimal policy is obtained after each episode by the leaning agent which employing a Q-learning then update the Q-value table. The sub optimal policy for each scheduler agent is estimated by the neural network model and then obtains the optimal policy for the whole situation which created by the global reward (central learner agent's reward) and local rewards vector. The learning agent continue update the Q-value table until the leaning process is finished which can balances the process of exploration and exploitation efficiently.

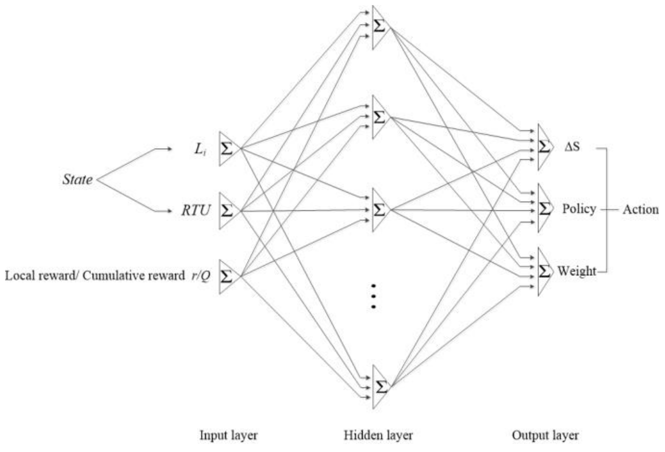


Figure 5. The decision making in the proposed system.

Each scheduler agent places its local rewards in a vector and sends it to the learner agent (Figure 5). As mentioned, the leaner agent receives a set of local rewards produced by the scheduler agents in each step. The dynamic changes can be represented by a function of the relation between this vector of local rewards and the current resource total usage as:

$$\Delta S = f(R_{loc}, RTU) \tag{13}$$

Where ΔS is the resource change (dynamicity), R_{loc} is the local rewards, and RTU is the resource total usage (including the queue remainder buffer size).

The policy estimation in decision making can be represented by the following scenario:

a) The global leaner agent (after receiving the local rewards) is responsible of sending the new Q-table after been updated to the all scheduler agents.
b) The global Q-table can be represented by a vector that its size created by the number of all scheduler agents.
c) The central learning agent (global agent) uses the local rewards vector to updates the Q-table as:

$$G(q) = (1-\alpha) * G(q) + \alpha * \sum_i RV_i(q) \tag{14}$$

Where *G(q)* is the global Q-value, α is the learning factor, $RV_i\ (q)$ is the ith Q-value in the local rewards vector created by the ith scheduler agent for the ith resource. The ith resource efficiency is evaluated by the summation of all related rewards.

The decision making process is built on choosing the best action to schedule the current job based on the aforementioned parameters. Therefore, the best action would be taken by the learning agent according to the highest estimated value for ΔS function. Each scheduling agent takes a job from its queue and selects the suitable resource which has the highest estimated value in the global Q-table and submits the job to it. As mentioned, only the global learning agent is responsible of collecting the local rewards, updates the global Q-table and sends the new Q-value to the scheduler agents. However, to avoid the single-point failure occurrence in our system, we designed flexibility in holding the global learning agent job position. Any one of the scheduler agents can hold the global learning agent position and hold its role in case of failure occurrence.

6. Experimental Results

To evaluate our proposed approach TSMRL performance we conducted several experiments through a developed simulation. We compared the performance of our approach with other scheduling algorithms such as First In First Serve (FIFS), Greedy and Random scheduling algorithms in terms of successful job execution, response time, utilization and Average Load of Resources (AloR). In the FIFS scheduling method, the scheduler agent selects resource in a fair way and the resources selection is not considered under any rules unless the priority of resource allocation FIFS. In the Greedy method, the scheduler agent schedules the jobs in greedy approach which is actually similar to the reinforcement learning in some concepts. In which the scheduler in Greedy algorithm search for optimum scheduling locally then decide which next step of scheduling will maximize the benefits. In Random method, the scheduler agent randomly selects the resources and schedules the jobs without any consideration for load or resource total usage. The experiments setup and parameters are stated in Table 1.

Table 1. Parameters and values.

Parameter	Value
VM buffer	10-50
PEs	5
Jobs total number	100-1000
VM memory (RAM) mega byte	512-2048
VMs total number	10
VM frequency (Million Instructions Per Second)	1-30
Job length (million instructions)	100-2000
Bandwidth (Million Bits Per Second)	1-2

In Figure 6 we conducted a comparison between our approach TSMRL and the other methods in terms of response time under different arrival rate (from 10 to 20 user requests per second). As we can see in the figure, the proposed approach demonstrated a lowest response time which indicate to the efficiency and reliability enhancement by using the proposed approach.

In Figure 7 a comparison between the proposed approach and the other approach in term of successfully scheduled jobs is plotted under different deadlines varies from 10 to 60 minutes. In the figure we can notice that the successful jobs are increase with the dead line incremental which is obvious in the proposed approach the successful jobs incremental rate is higher compared with the other approach.

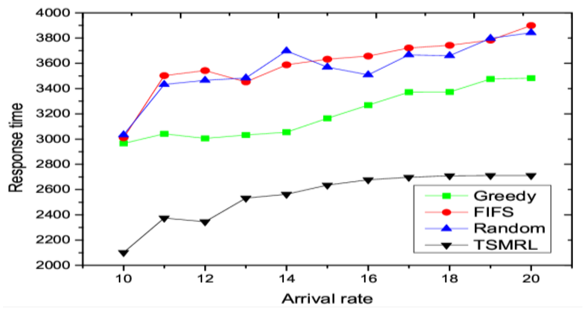


Figure 6. The response time under different arrival rate (request/per second).

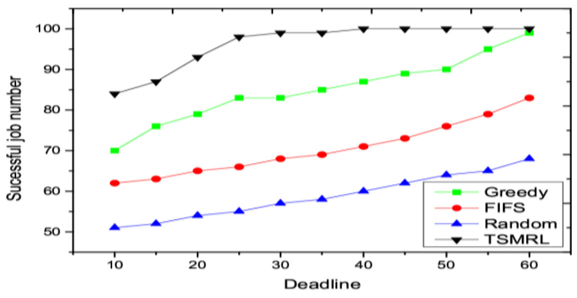


Figure 7. The successful jobs number under varies deadline.

The evaluation metric AloR has been used for perfomance evaluation in grid and cloud computing job scheduling because AloR can efficiency assess the job scheduling performance in grid and cloud environment [30]. As we consider the resources heterogeneity in our proposed model, the processing power (capacity) for each resouce is different from the other. The processing capacity for each resource is defined as the inverse of the required CPU time to perform a unit of a job length [30]. Taking the above scheduling methods (Greedy, FIFS, Random, and TSMRL) in consideration, each resource or VM has a queue to receive the arriving user requests in it which perform only one job at a time based on the adopted scheduling method. In addition, each job has its own length which needs to be performed in a given interval. Taking those assumptions in consideration, we can calculate ALoR as follows:

$$ALoR = (\frac{1}{|N|}) * \sum_r LoR_r = (\frac{1}{|N|}) * \sum_r (\frac{L_r}{C_r}) \tag{15}$$

Where |N| is the number of resources, LoR_r is the load of r^{th} resouce. L_r is the queue total jobs length for the r^{th} resouce, C_r is the capacity of r^{th} resouce.

The lower evaluation value of metric ALoR is means better performance because it is indecate to beter load balancing among the resources. Alternatively, higher evaluation value for ALoR means unsatisfied performance for the same reason of lower load balancing level. Evaluation of our proposed approach has been done in two different scales (small and medium) and two levels of loads (medium and heavy). The number of resources and schedulers defines the system scale while the load of the system is driven from the proportion of total jobs length in the queue buffer to the capacity of related resource.

In all figures, the performance of our proposed approach TSMRL is lower (higher ALoR) in the initial steps as the learner system needs to acquire history from the past experiences. Alternatively, our proposed method performs better (lower ALoR) as there is incremental in the steps because of the gained information from the past experiments. The parameters of scales and load for the experiments have been plotted as follows: Figure 8 small scale (30 schedulers) and medium load (60%); Figure 9 large scale (60 schedulers) and medium load (60%); Figure 10 small scale (30 schedulers) and heavy load (90%); Figure 11 large scale (60 schedulers) and heavy load (90%).

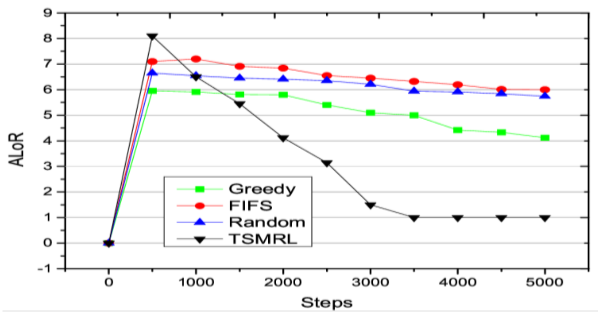


Figure 8. ALoR small scale and medium load.

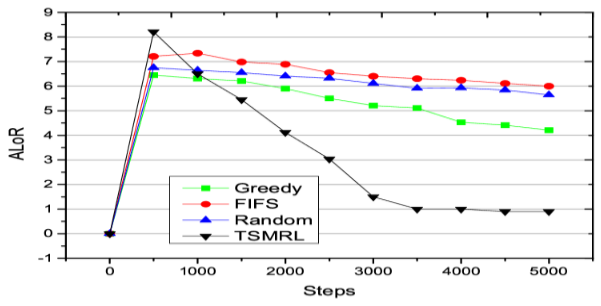


Figure 9. Large scale and medium load.

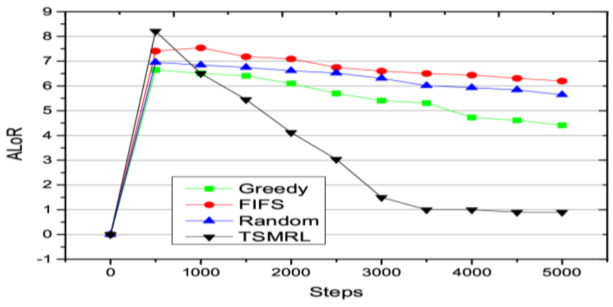


Figure 10. ALoR small scale and heavy load.

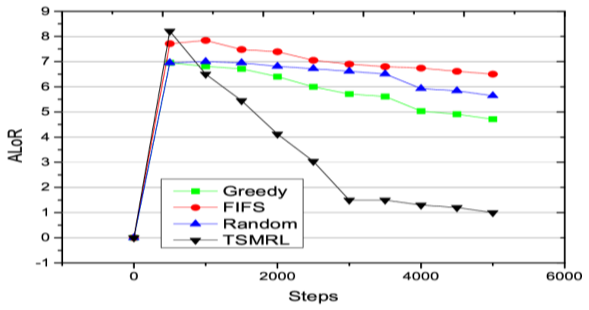


Figure 11. Large scale and heavy load.

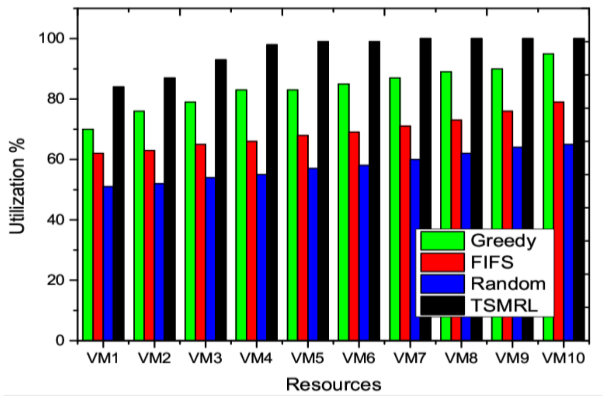


Figure 12. The utilization rate for 10 virtual machines.

In Figure 12, utilization rate of 10 virtual machines is plotted. As we can noticed from the figure, our proposed approach TSMRL achieved the best utilization rate among the other approaches which is shoes the efficiency of our proposed approach.

7. Conclusions

In this paper, we developed a multi-agent task scheduler to enhance the reliability in cloud environmment. In our proposed model, we impmelemnted NFQ-based Reinforcement learning algorithm to efficiently schedule the task and utilize the resources in a proper way. The proposed model contains several scheduler agents which are responsible of task scheduling with resource total usage and queue buffer size consideration. In our approach we implemented neural network as a support for reinforcement learning algorithm to learn and solve the problem of the dynamicity in cloud environment. Each scheduler agent reserve the user requests and schedules it based on the resources status in the designated moment then sends the reward as a Q-value to the global learning agent. The global learning agent evaluate the taken action through the past experience and sends a Q-value back to the scheduler agents to take the next action in the next step.

References

[1] Abdallah S. and Lesser V., "Learning the Task Allocation Game," *in Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, Hakodate, pp. 850-857, 2006.

[2] Bahrpeyma F., Golchin B., and Cranganu C., "Fast Fuzzy Modeling Method to Estimate Missing Logsin Hydrocarbon Reservoirs," *Journal of Petroleum Science and Engineering*, vol. 112, pp. 310-321, 2013.

[3] Bahrpeyma F., Zakerolhoseini A., and Haghighi H., "Using IDS Fitted Q to Develop A Real-Time Adaptive Controller for Dynamic Resource Provisioning in Cloud's Virtualized Environment," *Applied Soft Computing*, vol. 26, pp. 285-298, 2015.

[4] Bu X., Rao J., and Xu C., "A Reinforcement Learning Approach to Online Web Systems Auto-Configuration," *in Proceedings of International Conference on Distributed Computing Systems*, Montreal, pp. 2-11, 2009.

[5] Chang R., Chang J., and Lin P., "An Ant Algorithm for Balanced Job Scheduling in Grids," *Future Generation Computer Systems*, vol. 25, no. 1, pp. 20-27, 2009.

[6] Ayyapazham R. and Velautham K., "Proficient Decision Making on Virtual Machine Creation in IaaS Cloud Environment," *The International Arab Journal of Information Technology*, vol. 14, no. 3, pp. 314-323, 2017.

[7] Cirne W. and Berman F., "When the Herd is Smart: Aggregate Behavior in the Selection of Job Request," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 2, pp. 181-192, 2003.

[8] Farahnakian F., Liljeberg P., and Plosila J., "Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers Using Reinforcement Learning," *in Proceedings of 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Torino, pp. 500-507, 2014.

[9] García S., Prado R., and Expósito J., "Fuzzy Scheduling With Swarm Intelligence-Based Knowledge Acquisition for Grid Computing," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 2, pp. 359-375, 2012.

[10] Gabel T., Lutz C., and Riedmiller M., "Improved Neural Fitted Q Iteration Applied to A Novel Computer Gaming and Learning Benchmark," *in Proceedings of Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, Paris, pp. 279-286, 2011.

[11] Guo Y., Lama P., Jiang C., and Zhou X., "Automated and Agile Server Parametertuning by Coordinated Learning and Control," *Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, pp. 876-886, 2014.

[12] Huang Y., Bessis N., Norrington P., Kuonen P., and Hirsbrunner B., "Exploring Decentralized Dynamic Scheduling for Grids and Clouds Using The Community-Aware Scheduling Algorithm," *Future Generation Computer S*ystems, vol. 29, no. 1, pp. 402-415, 2013.

[13] Hussin M., Hamid N., and Kasmiran K., "Improving Reliability in Resource Management through Adaptive Reinforcement Learning for Distributed Systems," *Journal of parallel and Distributed Computing*, vol. 75, pp. 93-100, 2015.

[14] Ilg W., Berns K., Mühlfriedel T., and Dillmann R., "Hybrid Learning Concepts Based on Self-Organizing Neural Networks for Adaptive Control of Walking Machines," *Robotics and Autonomous*

Systems, vol. 22, no. 3-4, pp. 317-327, 1997.

[15] Khan S., Herrmann G., Lewis F., Pipe T., and Melhuish C., "Reinforcement Learning and Optimal Adaptive Control: An Overview and Implementation Examples," *Annual Reviews in Control*, vol. 36, no. 1, pp. 42-59, 2012.

[16] Khazaei H., Misic J., and Misic V., "A Fine-Grained Performance Model of Cloud Computing Centers," *Transactions on parallel and distributed systems*, vol. 24, no. 11, pp. 2138-2147, 2013.

[17] Krauter K., Buyya R., and Maheswaran M., "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," *Software: Practice and Experience*, vol. 32, no. 2, pp. 135-164, 2002.

[18] Lin Y. and Li X., "Reinforcement Learning Based on Local State Feature Learning and Policy Adjustment," *Information Sciences*, vol. 154, no. 1-2, pp. 59-70, 2003.

[19] Liu X., Tong W., Zhi X., ZhiRen F., and WenZhao L., "Performance Analysis of Cloud Computing Services Considering Resources Sharing Among Virtual Machines," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 357-374, 2014.

[20] Llorente I., Moreno R., and Montero R., "Cloud Computing for On-Demand Grid Resource Provisioning," *Advances in Parallel Computing*, vol. 18, pp. 177-191, 2009.

[21] Pauli S., Kohler M., and Arbenz P., "A fault tolerant implementation of Multi-Level Monte Carlo methods," *Parallel computing: Accelerating computational science and Engineering*, vol. 25, pp. 471-480, 2014.

[22] Riedmiller M., "Neural Fitted Q Iteration-First Experiences with A Data Efficient Neural Reinforcement Learning Method," *in Proceedings of European Conference on Machine Learning*, Porto, pp. 317-328, 2005.

[23] Rizvandi N., Taheri J., Moraveji R., and Zomaya A., "A Study on Using Uncertain Time Series Matching Algorithms for Mapreduce Applications," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1699-1718, 2013.

[24] Sutton R. and Barto A., *Reinforcement Learning: An introduction*, MIT press, 2018.

[25] Tesauro G., "Practical Issues in Temporal Difference Learning," *Machine Learning*, vol. 8, pp. 257-277, 1992.

[26] Tesauro G., Jong N., Das R., and Bennani M., "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation," *in Proceedings of the 3rd International Conference on Autonomic Computing*, Dublin, pp. 65-73, 2006.

[27] Vengerov D., "A Reinforcement Learning Approach to Dynamic Resource Allocation," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 3, pp. 383-390, 2007.

[28] Vishkin U., Caragea G., and Lee B., *Models, Algorithms and Applications, Chapter Models for Advancing PRAM and Other Algorithms Into Parallel Programs for A PRAM-On-Chip Platform*, Handbook of Parallel Computing CRC Press, 2006.

[29] Watkins C., "Learning from Delayed Rewards," PhD Thesis, King's College, 1989.

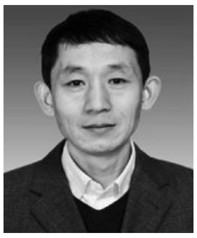
[30] Wu J., Xu X., Zhang P., and Liu C., "A Novel Multi-Agent Reinforcement Learning Approach for Job Scheduling in Grid Computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 430-439, 2011.

[31] Xhafa F. and Abraham A., "Computational Models and Heuristic Methods for Grid Scheduling Problems," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 608-621, 2010.

[32] Zhang C., Lesser V., and Shenoy P., "A Multi-Agent Learning Approach to Resource Sharing Across Computing Clusters," UMass Computer Science Technical Report, University of Massachusetts Amherst, 2008.

[33] Zheng Q., Yang H., and Sun Y., "How to Avoid Herd: A Novel Stochastic Algorithm in Grid Scheduling," *in Proceedings of 15th IEEE International Conference on High Performance Distributed Computing*, Paris, pp. 267-278, 2006.

Husamelddin Balla Received his MSc in Computer Science from Harbin Institute of Technology. He is a research scholar at Northeast Forestry University. His research interests include cloud computing, machine learning and natural language processing.



Chen Sheng is currently a Doctoral Supervisor and a Professor with Northeast Forestry University, China. He is also a member of the National Innovation Methods Research Institute and the Executive Director of the Education Information Technology Council of Education Ministry. He has published over 30 academic papers and one monograph. His research interests include biomass material prediction, intelligent detection of new composite materials, and big data on forestry.

Jing Weipeng received the Ph.D. degree from the Harbin Institute of Technology, China. He is currently an Associate Professor with Northeast Forestry University, China. He has published over 50 research articles in refereed journals and conference proceedings, such as CPC, PUC, and FGCS. His research interests include modeling and scheduling for distributed computing systems, fault tolerant computing and system reliability, cloud computing, and spatial data mining.