Adaptive and Efficient Resource Allocation in Cloud Datacenters Using Actor-Critic Deep Reinforcement Learning

Zheyi Chen, Jia Hu, Geyong Min, Chunbo Luo, and Tarek El-Ghazawi, *Fellow, IEEE*

Abstract—The ever-expanding scale of cloud datacenters necessitates automated resource provisioning to best meet the requirements of low latency and high energy-efficiency. However, due to the dynamic system states and various user demands, efficient resource allocation in cloud faces huge challenges. Most of the existing solutions for cloud resource allocation cannot effectively handle the dynamic cloud environments because they depend on the prior knowledge of a cloud system, which may lead to excessive energy consumption and degraded Quality-of-Service (QoS). To address this problem, we propose an adaptive and efficient cloud resource allocation scheme based on Actor-Critic Deep Reinforcement Learning (DRL). First, the actor parameterizes the policy (allocating resources) and chooses actions (scheduling jobs) based on the scores assessed by the critic (evaluating actions). Next, the resource allocation policy is updated by using gradient ascent while the variance of policy gradient is reduced with an advantage function, which improves the training efficiency of the proposed method. We conduct extensive simulation experiments using real-world data from Google cloud datacenters. The results show that our method can obtain the superior QoS in terms of latency and job dismissing rate with enhanced energy-efficiency, compared to two advanced DRL-based and five classic cloud resource allocation methods.

Index Terms—Cloud computing, datacenters, resource allocation, energy-efficiency, deep reinforcement learning

1 Introduction

Cloud computing has rapidly developed as one of the most prevailing computing paradigms [2]. In cloud computing, resource allocation is regarded as a process of allocating computing, storage, and networking resources to meet the requirements of both users and cloud service providers (CSPs). Many problems in cloud resource allocation have emerged with the ever-increasing scale and dynamics of cloud datacenters, such as irrational resource provisioning and slow response to changes. These problems not only degrade the Quality-of-Service (QoS) but also cause high energy consumption and maintenance overheads [3]. Therefore, it has been a high-priority objective to design an adaptive and efficient solution for resource allocation in cloud datacenters. However, it is a highly challenging task due to the dynamic system states and various user demands in cloud computing [4], as described below.

- *The complexity of cloud datacenters.* There are a large number of different types of servers in cloud datacenters, which provide various computing and storage resources including central processing units (CPUs), memories, and storage units. Therefore, it is challenging to manage and coordinate such heterogeneous resources efficiently in cloud computing [5].
- *The diversity of demands from users.* Jobs coming from different users demand heterogeneous resources (e.g., CPUs, memories, and storage units) and different durations (e.g., minutes, hours, and days) [6]. Such diversity of user demands intensifies the difficulty of resource allocation in cloud datacenters.
- *The excessiveness of energy consumption.* Large energy consumption not only causes huge operation overheads but also results in extensive carbon emissions [7]. In Google cloud datacenters, the average CPU utilization of servers is only around 20% [8]. Such energy waste occurs when irrational resource allocation schemes are used. However, it is hard to satisfy diverse user demands while maintaining cloud datacenters with high energy-efficiency.
- *The dynamics of cloud systems.* In cloud datacenters, system states such as resource usage and requests are changing frequently. Effective resource allocation is expected to continuously meet the requirements of user jobs under such dynamic cloud environments. However, it is difficult to build an accurate model for resource allocation in response to dynamic cloud environments. Therefore, these dynamics have caused huge challenges to adaptive resource allocation in cloud datacenters [9].

- *Zheyi Chen, Jia Hu, Geyong Min, and Chunbo Luo are with the Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, EX4 4QF Exeter, U.K. E-mail: {zc300, j.hu, g.min, c.luo}@exeter.ac.uk.*
- *Tarek El-Ghazawi is with the Department of Electrical and Computer Engineering, The George Washington University, Washington, DC 20052 USA. E-mail: tarek@gwu.edu.*

Manuscript received 5 Aug. 2020; revised 19 Oct. 2021; accepted 28 Nov. 2021. Date of publication 3 Dec. 2021; date of current version 15 Dec. 2021.
This work was supported in part by the EU Horizon 2020 INITIATE project under Grant 101008297. The preliminary version of this work appeared in IEEE ICC 2019 [1].
(Corresponding authors: Jia Hu and Geyong Min.)
Recommended for acceptance by J. Wang.
Digital Object Identifier no. 10.1109/TPDS.2021.3132422

Many classic solutions for cloud resource allocation are based on rules [10], heuristics [11], and control theory [12]. Although these solutions can solve the problem of cloud resource allocation to some extent, they commonly use the prior knowledge of cloud systems (e.g., state transitions, demand changes, and energy consumptions) to develop corresponding strategies of resource allocation. Thus, these solutions might work well in a specific application scenario, but they are unable to fully fit in the cloud environment with dynamic system states and user demands. For example, job scheduling can be easily executed by using rule-based strategies for meeting instant user demands. However, they only consider the current job characteristics (e.g., resource demands and job durations) to obtain short-term benefits. Therefore, they are unable to adaptively fulfill the dynamic demands of user jobs with a long-term perspective, and it might result in excessive job latency and serious resource wastes due to irrational resource allocation. Besides, numerous iterations may be needed to find feasible resource allocation plans with these solutions, which leads to high computational complexity and resource overheads. Therefore, they are unable to effectively address the complicated problem of resource allocation in dynamic cloud environments.

Reinforcement learning (RL) [13] has emerged as a promising approach for handling resource allocation problems with high-adaptiveness and low-complexity. However, traditional RL-based methods suffer from the problem of high-dimensional state space when dealing with complex cloud environments [14]. To address this problem, deep reinforcement learning (DRL) [15] was proposed to extract low-dimensional representations from high-dimensional state spaces using deep neural networks (DNNs) [16]. Although there are some DRL-based methods focused on the problem of cloud resource allocation [17], [18], [19], [20], most of them use the value-based DRL (e.g., deep Q-networks (DQN) [15] and double Q-learning (DQL) [21]), which may lead to low training efficiency when dealing with a large action space. This is because the value-based DRL learns a deterministic policy by calculating the probability of each action. However, in a cloud datacenter, jobs may arrive constantly and thus the action space may be considerably large to continuously meet the requirements of scheduling jobs. Therefore, it could be hard for the value-based DRL to approach the optimal policy with quick convergence. By contrast, the policy-based DRL (e.g., policy gradient (PG) [13]) learns a stochastic policy and can better deal with the large action space in a cloud datacenter by directly outputting actions with the probability distribution, but it might reduce the training efficiency caused by the high variance generated when estimating the policy gradient.

As a synergy of value-based and policy-based DRL algorithms, advantage actor-critic (A2C) [13] was designed to address the above issues. In A2C, the actor chooses actions based on the scores assessed by the critic, where the variance of policy gradient is reduced with an advantage function. However, the A2C adopts a single-thread training manner and thus underutilizes computational resources. Meanwhile, strong data correlation may occur when using the A2C because similar training samples are generated when there is only a single DRL agent interacting with the environment, which would cause unsatisfactory training results. To address these problems with A2C, an asynchronous advantage actor-critic (A3C) algorithm with low-variance and high-efficiency was proposed in [22]. The A3C uses multiple DRL agents to interact with the environment simultaneously, making full use of computational resources and thus improving the learning speed. Meanwhile, the data collected by different DRL agents are independent of each other, and thus the A3C breaks the data correlation.

In light of the A3C algorithm's advantages, we develop an A3C-based resource allocation scheme for cloud datacenters with heterogeneous resources, diverse user demands, large energy consumption, and dynamic environments. The main contributions of this paper are summarized as follows.

- A unified model of resource allocation is designed for a cloud datacenter with dynamic system states and heterogeneous user demands. In the proposed model, the QoS (job latency and dismissing rate) and energy-efficiency (average energy consumption of jobs) are regarded as optimization goals. Furthermore, the state space, action space, and reward function for cloud resource allocation are defined and formulated as a Markov decision process (MDP), which are used in the proposed DRL-based cloud resource allocation scheme.
- An actor-critic DRL (A3C) based resource allocation method is proposed to efficiently approach the optimal policy of job scheduling in a cloud datacenter. Specifically, DNNs are utilized to handle the problem of high-dimensional state space in a cloud datacenter. Moreover, the training efficiency of the proposed method is greatly improved with the asynchronous update of policy parameters among multiple DRL agents.
- The extensive simulation experiments using real-world trace data from Google cloud datacenters are conducted to validate the effectiveness of the proposed method. The simulation results demonstrate that the proposed method can achieve the better QoS, higher energy-efficiency, and faster convergence compared to five classic resource allocation algorithms and two advanced DRL-based resource allocation methods.

The rest of this paper is organized as follows. In Section 2, the related work is introduced. Section 3 describes the system model of resource allocation in a cloud datacenter. In Section 4, the proposed A3C-based cloud resource allocation method is presented in detail. In Section 5, the proposed method is evaluated by simulation experiments with real-world datasets. Section 6 concludes this paper.

2 Related Work

Resource allocation in cloud computing has attracted much research attention, while many studies have contributed to solving this important problem. In this section, we review the related work from two aspects, including the classic and DRL-based solutions for cloud resource allocation.

2.1 Classic Approaches for Cloud Resource Allocation

Resource allocation problem is omnipresent in cloud computing and many methods have been proposed to enhance resource utilization with rational resource provisioning and effective cost control. For example, Zahid *et al.* [10] proposed a ruled-based language for CSPs, in order to improve the QoS compliance of high-performance computing (HPC) clouds. Through using probabilistic thresholds, a system model was designed in [23] for accomplishing the switching between different operating levels of cloud services. Johnson's rule and genetic algorithm were combined in [24] for solving the multiprocessor scheduling problem in cloud datacenters. By using a rule-based control approach, a power-aware job scheduler was designed in [25] for improving power-efficiency and meeting power constraints. Wang *et al.* [25] also compared the pros and cons of baseline scheduling algorithms, such as the longest job first (LJF), shortest job first (SJF), and round-robin (RR). Furthermore, Samal and Mishra [26] analyzed the variants of RR algorithms for load balancing in cloud computing. Based on the heuristics, the problem of cloud resource reservation was solved in [11] for meeting user demands while reducing resource costs. Grandl *et al.* [27] designed Tetris, a cluster scheduler with packing heuristics, in order to match task requirements with resource availability and improve cluster efficiency. Avgeris *et al.* [12] proposed a hierarchical resource allocation framework with an admission control mechanism, and it can support mobile users to choose edge servers for executing their tasks with less response time and computational costs. Haratian *et al.* [28] designed an adaptive resource management framework for meeting the QoS requirements, where the decision-making of cloud resource allocation was executed by a fuzzy controller in each iteration of control cycles. Through utilizing the feedback-control theory, a Big-Data MapReduce system was developed in [29] for reducing the costs of cluster reconfigurations.

Overall, most of these work focused on rule-based strategies, heuristics, and control theory for cloud resource allocation. The rule-based strategies or heuristics need to establish different rules for fulfilling dynamic system states and user demands in cloud datacenters. Thus, not only the application scopes of them are limited but also high overheads of rule settings are generated. Besides, the control-theory based solutions require numerous feedback iterations, and thus they usually lead to high computational complexity and unnecessary resource overheads. To address these important challenges, DRL has emerged as an adaptive and efficient decision-making method for solving the complicated problem of cloud resource allocation.

2.2 DRL-Based Methods for Cloud Resource Allocation

Deep reinforcement learning (DRL) combines reinforcement learning (RL) and DNNs, and it emphasizes the decision-making process of choosing actions according to different system states in the dynamic environments, in order to maximize the long-term rewards. Two great milestones have witnessed the vigorous development of DRL-based algorithms. One is the application of deep Q-networks (DQN) on the Atari 2600 platform [15]. The other is the Alpha Go that defeated the Go world champion by integrating Monte Carlo Tree Search (MCTS) with DNNs [30]. Moreover, DRL-based methods have recently been applied to the problem of resource allocation in cloud computing. For instance, Tong *et al.* [17] combined the Q-learning algorithm with DNNs to handle the scheduling problem of directed acyclic graph (DAG) tasks in the cloud environment. The DQN algorithm was adopted in [18] to allocate compute-intensive jobs, in order to reduce the energy consumption of cloud datacenters. By using the DQN algorithm, a hierarchical framework was designed in [19] for adaptive resource allocation, aiming to reduce power consumption in cloud datacenters. Also, Zhang *et al.* [31] adopted the DQN algorithm, in order to achieve adaptive provisioning and configuration for cloud resources. Different from these work using value-based DRL algorithms, Mao *et al.* [20] leveraged a policy-based DRL algorithm (i.e., policy gradient (PG)) to handle the resource allocation problem in cloud datacenters. Subject to resource constraints in wireless systems, Eisen *et al.* [32] utilized a PG-based method to find the near-optimality of resource allocation. Based on the PG algorithm, a QoS-aware scheduler was developed in [33], aiming to improve the QoS when scheduling DNN inference workloads in cloud computing. Moreover, a PG-based actor-critic approach for user scheduling and resource allocation was designed in [34], aiming to maximize the energy-efficiency in heterogeneous networks. Besides, an actor-critic based DRL method for cloud resource allocation was developed in our previous work [1] that only considered the optimization of job latency but not energy-efficiency. Moreover, the training efficiency of these actor-critic based DRL methods can still be improved because they did not take advantage of asynchronous update mechanism.

In general, most of these work depends on value-based DRL methods for cloud resource allocation. It is difficult for them to approach the accurate optimal policy when dealing with a large action space. Although there exists a small amount of research using policy-based DRL methods to address this problem, the high variance is generated when they estimate the policy gradient. Besides, the above DRL-based methods reveal drawbacks in training efficiency, thus numerous iterations are needed for optimizing the scheduling policy. To address these essential problems, we propose an A3C-based resource allocation method in cloud datacenters. Different from the GA3C [35] that aims to enhance the performance of the A3C algorithm by leveraging the GPU computational power, our method focuses on the adaptation and application of the A3C algorithm in cloud resource allocation.

3 System Model

A unified model of resource allocation is designed, aiming to improve the QoS and energy-efficiency in dynamic environments of cloud datacenters with various user demands and ever-changing system states. For the clarity of presentation, we consider the scenario of a single cloud datacenter with a set of servers, donated by $V = \{v_1, v_2, \dots, v_m\}$, where m indicates the number of servers. Each server provides multiple types of resources (e.g., CPUs, memories, and storage units), donated by $Res = \{r_1, r_2, \dots, r_n\}$, where n indicates the number of resource types. As shown in Fig. 1, a

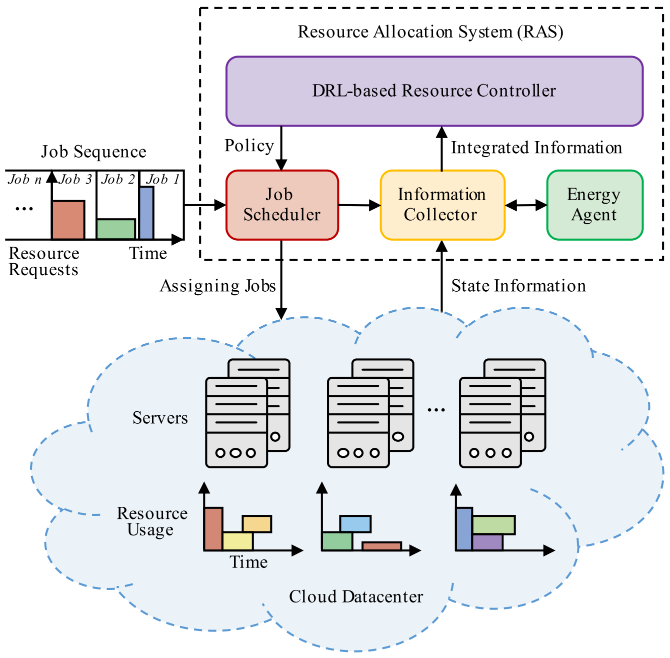


Fig. 1. Model of resource allocation in a cloud datacenter.

DRL-based resource controller is embedded in the resource allocation system (RAS). The RAS generates policies of job scheduling based on the resource requests of different user jobs and current state information of the cloud datacenter (e.g., number of servers, resource usage, and energy consumption). According to the policies delivered by the DRL-based resource controller, the job scheduler assigns jobs from the job sequence to servers. Specifically, the jobs are generalized as data processing jobs [36], such as the training jobs of deep learning (DL) models for image processing and speech recognition. For different jobs, they exhibit various resource requests according to their purposes. Therefore, each job consists of a specific job duration (e.g., minutes, hours, or days) and the request for different types of resources (e.g., CPUs and memories). During the process of resource allocation, the information collector records the usage of different resources and current energy consumption (measured by an energy agent) in the cloud datacenter. Referring to the above information, the DRL-based resource controller will generate policies of job scheduling accordingly. The major notations involved in the proposed model are listed in Table 1.

Considering that there are a set of all jobs that are expected to be processed, denoted by $J_{total} = \{j_1, j_2, \ldots, j_p\}$, where p indicates the total number of jobs, a set of jobs that are waiting in the job sequence, denoted by $J_{seq} = \{j_1, j_2, \ldots, j_q\}$, where q indicates the number of jobs waiting in the job sequence, and $q \leq p$. When a job from J_{total} arrives, it will first enter J_{seq}. If the available resources are enough, this job can be processed immediately. Otherwise, this job will wait in the job sequence for scheduling. Following the first-in-first-out (FIFO) policy, the jobs in J_{seq} will be dropped when the job sequence is full. Therefore, the actual completion time of a job is obtained by calculating the time interval from entering the job sequence to the end of processing, denoted by $T^j_{finish} - T^j_{enter}$. Fig. 2 illustrates an example of job scheduling. We assume that there is a server with 100 computing units of CPU resources. The jobs j_1, j_2, and j_3 request 50, 30, and 40 units of CPU resources, respectively, which arrive at timesteps t_1, t_2, and t_3, and are completed at timesteps t_4, t_5, and t_6. In the proposed model, the time instances (i.e., timesteps) are the arrival times and completion times of jobs at servers. Whenever a job arrives or is completed, a state transition occurs. More specifically, when j_1 and j_2 arrive, there are enough CPU resources to process these two jobs immediately. Therefore, the actual completion time of j_1 and j_2 (i.e., $t_4 - t_1$ and $t_5 - t_2$) are equal to their expected job durations (i.e., d_1 and d_2). However, j_3 can only be processed until j_1 is completed because the CPU resources are currently inadequate. Therefore, the actual completion time of j_3 (i.e., $t_6 - t_3$) is longer than its expected job duration (i.e., d_3).

TABLE 1
Major Notations Used in the Proposed Model

Notation	Definition
V	Set of servers in a cloud datacenter
Res	Types of resources in a cloud datacenter
J_{total}	Set of all jobs that are expected to be processed
J_{seq}	Set of jobs waiting in the job sequence
T^j_{finish}	Timestep when a job is finished
T^j_{enter}	Timestep when a job enters the job sequence
L_{normal}	Normalized average job latency
d_j	Duration of a job
$seqLen$	Length of job sequence
$disRate$	Job dismissing rate
E_{total}	Total energy consumption of a cloud datacenter
k	Fraction of energy consumption for an idle server
P_{max}	Maximum energy consumption of a server
U^{res}_t	Resource usage of all servers by timestep t
E_{job}	Average energy consumption of completed jobs
s_t	State of a cloud datacenter by timestep t
O^{res}_t	Occupancy request of all arrived jobs by timestep t
D^{job}_t	Durations of all arrived jobs by timestep t
a_t	Action adopted by job scheduler at timestep t
R_t	Total rewards at timestep t
R^{QoS}_t	Rewards of the QoS at timestep t
R^{energy}_t	Rewards of the energy-efficiency at timestep t
$T^{j,wait}_t$	Waiting time of job j by timestep t
$T^{j,work}_t$	Execution time of job j by timestep t
$T^{j,miss}_t$	Time consumption when job j is dismissed by timestep t
$E^{j,exec}_t$	Energy consumption of executing job j by timestep t

The numerical discrepancy among different values of job latency may lead to excessive computation time during gradient descent, therefore, the normalization is used to improve the training speed and convergence of the algorithm. Therefore, L_{normal} is defined as the normalized average job latency, which normalizes the job latency of all successfully completed jobs and then takes their average.

$$L_{normal} = \frac{\sum_{j \in completed\ jobs}\left(\left(T^j_{finish} - T^j_{enter}\right)/d_j\right)}{Number\ of\ completed\ jobs}, \tag{1}$$

where $L_{normal} \geq 1$ and d_j is the duration of a job.

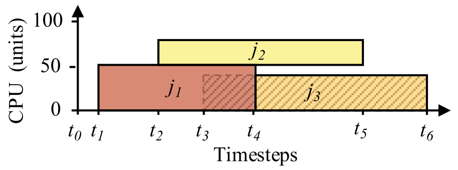


Fig. 2. An example of job scheduling.

Moreover, a constraint is added on the length of the job sequence J_{seq}, denoted by $seqLen$, which is used to avoid the QoS degrading caused by the excessive number of jobs that are staying in the waiting status. Therefore, $disRate$ is defined as the job dismissing rate, which calculates the rate of dismissed jobs when the job sequence is full.

$$disRate = 1 - \frac{Number\ of\ completed\ jobs}{Total\ number\ of\ jobs}, \tag{2}$$

where $0 \leq disRate \leq 1$.

Besides, the energy consumption generated in cloud datacenters commonly depends on their resource usage, and thus it is a feasible way to reduce energy consumption by enhancing this metric. This is because fewer servers tend to be switched on when the existing servers have high resource usage. Through experimental measurements, existing studies [19], [37], [38] have shown that the energy consumption of a server is proportional to its resource usage. Based on these studies, the total energy consumption of a cloud datacenter is formulated as

$$E_{total} = \sum_{t=0}^{T} \sum_{v \in V} \left(k \cdot P_{max} + (1-k) \cdot P_{max} \cdot U_t^{res} \right), \tag{3}$$

where P_{max} is the maximum energy consumption of a server when it is fully utilized, the fraction k is used to calculate the energy consumption of an idle server, U_t^{res} is the resource usage of all servers by timestep t, and T is the timestep when the last job is completed.

Different from most of the existing work that regards the total energy consumption as a performance metric, we consider the energy-efficiency during the job scheduling process, which is measured by the average energy consumption of all successfully completed jobs as

$$E_{job} = \frac{E_{total}}{Number\ of\ completed\ jobs}. \tag{4}$$

To improve the QoS (i.e., L_{normal} and $disRate$) and energy-efficiency (i.e., E_{job}), a DRL-based resource allocation method is proposed to execute the job scheduling in a cloud datacenter. Specifically, we regard the RAS as a DRL agent and the cloud datacenter as the environment. At each timestep, the DRL agent chooses an action of scheduling jobs by interacting with the environment. Accordingly, the state space, action space, and reward function in DRL are defined as follows.

State space: In the state space S, the state $s_t \in S$ consists of the resource usage of all servers and resource requests of all arrived jobs by timestep t. On one hand, $U_t^{res} = [[u_{1,1}, u_{1,2}, \ldots, u_{1,n}], [u_{2,1}, u_{2,2}, \ldots, u_{2,n}], \ldots, [u_{m,1}, u_{m,2}, \ldots, u_{m,n}]]$ indicates the usage of different types of resources on all servers by timestep t, where $u_{m,n}$ is the usage of the nth resource type on the server v_m. On the other hand, $O_t^{res} = [[o_{1,1}, o_{1,2}, \ldots, o_{1,n}], [o_{2,1}, o_{2,2}, \ldots, o_{2,n}], \ldots, [o_{j,1}, o_{j,2}, \ldots, o_{j,n}]]$ indicates the occupancy requests of all arrived jobs for different types of resources by timestep t, where $o_{j,n}$ is the occupancy request of the latest arrived job j for the nth resource type, and $D_t^{job} = [d_1, d_2, \ldots, d_j]$ denotes the durations of all arrived jobs by timestep t. Therefore, the state of a cloud datacenter by timestep t is defined as

$$s_t = [s_t^V, s_t^J] = [U_t^{res}, [O_t^{res}, D_t^{job}]], \tag{5}$$

where $s_t^V = U_t^{res}$ and $s_t^J = [O_t^{res}, D_t^{job}]$ are used to represent the states of all servers and arrived jobs for the clarity of presentation. The state space changes when jobs arrive or are completed, and the dimension of the state space depends on the situation of servers and arrived jobs, calculated by $(mn + z(n+1))$, where m, n, and z are the number of servers, resource types, and arrived jobs, respectively.

Action space: At timestep t, the action a_t adopted by the job scheduler is to select and execute jobs from the job sequence, according to a policy of job scheduling delivered by the DRL-based resource controller. The policy is generated based on the current system state, and the job scheduler assigns jobs to a specific server for execution. Once a job is scheduled to an appropriate server, the server will automatically allocate corresponding resources according to the resource request of this job. Therefore, the action space only indicates whether a job will be processed by a server or not, which is defined as

$$A = \{a_t | a_t \in \{0, 1, 2, \ldots, m\}\}, \tag{6}$$

where $a_t \in A$. When $a_t = 0$, the job scheduler does not assign the job at timestep t and the job needs to wait in the job sequence. Otherwise, the job will be processed by a specific server.

State-transition probability matrix: The matrix indicates the probabilities of the transition between two states. Taking Fig. 2 as an example, at timestep t_0, there is no job to be processed and the initial state $s_0 = [0, [[0], [0]]]$, where the three "0" items represent the CPU usage of the server, occupancy requests of jobs, and job durations, respectively. At t_1, the job j_1 is scheduled immediately since the available resources are sufficient. After taking this action, the state evolves to $s_1 = [50, [[50], [d_1]]]$, where the first "50" item indicates the CPU usage of the server, the second "50" item represents the occupancy request of j_1 for CPU resources, and d_1 is the duration of j_1. Similarly, after taking the action of scheduling j_2 at t_2, the state evolves to $s_2 = [80, [[50, 30], [d_1, d_2]]]$. Specifically, the state-transition probability matrix is denoted as $\mathbb{P}(s_{t+1}|s_t, a_t)$, which indicates the probabilities of transiting to the next state s_{t+1} when taking an action a_t at the current state s_t. The values of the transition probabilities are obtained by running the DRL algorithm, which outputs the probabilities of taking different actions at a state.

Reward function: The reward function is used to guide the DRL agent (RAS) to learn better policies of job scheduling with higher discounted long-term rewards, aiming to improve the system performance of cloud resource allocation. Therefore, at timestep t, the total rewards R_t consist of two parts including the rewards of the QoS (denoted by R_t^{QoS}) and the energy-efficiency (denoted by R_t^{energy}), which is defined as

$$R_t = R_t^{QoS} + R_t^{energy}. \tag{7}$$

Specifically, R_t^{QoS} reflects the penalties (hence negative) for different types of latency at timestep t including $T_t^{j,wait}$,

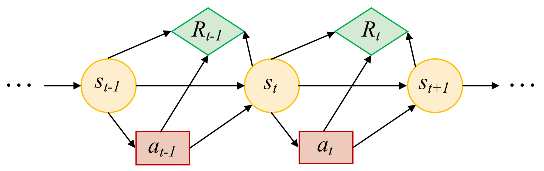


Fig. 3. An example of the MDP process modeling cloud resource allocation.

$T_t^{j,work}$, and $T_t^{j,miss}$ (as described in Table 1), which is defined as

$$R_t^{QoS} = -\sum_{j\in J_{seq}} \left(w_1 \cdot \frac{T_t^{j,wait} + T_t^{j,work}}{d_j} + w_2 \cdot T_t^{j,miss} \right), \tag{8}$$

where w_1 and w_2 are used to weight the penalties. Since R_t^{QoS} is a negative value, a job that has a longer duration tends to wait for a shorter time. This is sensible for a cloud system with the objective of profit maximization as a job with a longer duration can lead to higher profits [39].

Moreover, R_t^{energy} reflects the penalty for energy consumption at timestep t, which is defined as

$$R_t^{energy} = -w_3 \cdot \sum_{j\in J_{seq}} E_t^{j,exec}, \tag{9}$$

where $E_t^{j,exec}$ is the energy consumption of executing a job by timestep t, and w_3 is used to weight the penalty.

During the optimization process of cloud resource allocation, the DRL agent first chooses an action a_t (scheduling jobs) under the current system state s_t (resource usage and resource requests) of the environment (cloud datacenter). Next, the DRL agent receives rewards R_t (the QoS and energy-efficiency) and steps to the next state s_{t+1}. This process is illustrated by an MDP, as shown in Fig. 3.

Due to the uncertainty of system states, the problem of cloud resource allocation is formulated with model-free DRL. Based on the discrete-time-based MDPs with a large action space, an A3C algorithm is utilized to explore adaptive and efficient resource allocation in dynamic environments of cloud datacenters.

4 Adaptive and Efficient Resource Allocation Using A3C in Cloud Datacenters

This section presents the proposed effective resource allocation method based on the asynchronous advantage actor-critic (A3C), which can achieve superior QoS and energy-efficiency in cloud datacenters. The proposed method adopts an actor-critic based DRL framework with asynchronous update (A3C) to accelerate the training process. Specifically, the A3C-based method incorporates both value-based and policy-based DRL algorithms. On one hand, the value-based DRL determines the value function by using function approximators and adopts the ϵ-greedy to balance the exploration and exploitation. Therefore, the DRL agent utilizes existing experiences to choose good actions of job scheduling whilst exploring new actions. On the other hand, the policy-based DRL parameterizes the policy of job scheduling and directly outputs actions with probability distributions during the learning process without storing their Q-values. Thus, the DRL agent can efficiently choose actions under a large action space.

The key steps of the proposed A3C-based cloud resource allocation method are shown in Algorithm 1. Based on the definitions of state space (in Eq. (5)), action space (in Eq. (6)), and reward function (in Eq. (7)), the actor's network V^{π_θ} and critic's network Q^{π_θ} are first initialized with weights and biases. Next, the actor's and critic's learning rate γ_a and γ_c, and the TD error discount factor β are initialized.

Algorithm 1. The A3C-Based Resource Allocation in Cloud Datacenters

```
1  Initialize: The actor's network $V^{\pi_\theta}$ and critic's network $Q^{\pi_\theta}$
   with weights and biases.
2  Initialize: The actor's and critic's learning rate $\gamma_a$ and $\gamma_c$,
   reward decay rate $\lambda$, TD error discount factor $\beta$, counter
   $temp = 0$, and update step $u$.
3  for each training epoch $n = 0, 1, 2, \ldots, N$ do
4    Receive the initial state $s_0$, where $s_0 = env.observe()$;
5    for $t = 0, 1, 2, \ldots, T$ do
6      Select the action $a_t$ of scheduling jobs based on the
       current system state $s_t$ of the cloud datacenter, where
       $s_t = [s_t^V, s_t^J]$ (defined in Eq. (5)) and $a_t \in A$ (defined in
       Eq. (6)): $a_t = actor.choose\_action(s_t)$;
7      Execute the scheduling action $a_t$, receive the reward $R_t$
       (QoS and energy-efficiency) and the next state $s_{t+1}$,
       where $R_t = R_t^{QoS} + R_t^{energy}$ (defined in Eq. (7)):
       $R_t, s_{t+1} = env.step(a_t)$;
8      Calculate the discounted long-term rewards:
       $R_{disc} = R_0 + \lambda R_1 + \ldots + \lambda^{t-1} R_{t-1}$;
9      Calculate the advantage function in the critic, where
       $Q_w(s_t, a_t) = R_{disc} + \lambda^t V^{\pi_{\theta_t}}(s_{t+1})$:
       $A^{\pi_{\theta_t}}(s_t, a_t) = Q_w(s_t, a_t) - V^{\pi_{\theta_t}}(s_t)$;
10     Minimize the TD error:
       $\delta^{\pi_{\theta_t}} = R_t + \beta V^{\pi_{\theta_t}}(s_{t+1}) - V^{\pi_{\theta_t}}(s_t)$;
11     Update the state-action value function parameter:
       $w_{t+1} \leftarrow w_t + \gamma_c \delta_t^{\pi_{\theta_t}} \nabla_w Q_w(s_t, a_t)$;
12     Calculate the policy gradient in the actor by
       using the advantage function: $\nabla_{\theta_t} J(\theta_t) =$
       $E_{\pi_{\theta_t}}[\nabla_{\theta_t} log\pi_{\theta_t}(s_t, a_t) A^{\pi_{\theta_t}}(s_t, a_t)]$;
13     Update the scheduling policy: $\theta_{t+1} \leftarrow \theta_t + \gamma_a \nabla_{\theta_t} J(\theta_t)$;
14     Update the state: $s_t = s_{t+1}$;
15     Update the counter: $temp = temp + 1$;
16     if $temp \ \% \ u == 0$ then
17       Call Algorithm 2 to asynchronously update policy
         parameters in each DRL agent;
18     end
19   end
20 end
```

The optimization objective of the proposed A3C-based resource allocation method is to obtain the most rewards. Therefore, the instant reward R_t (defined in Eq. (7)) is accumulated by using a probability distribution as

$$J(\theta_t) = \sum_{s_t\in S} d^{\pi_{\theta_t}}(s_t) \sum_{a_t\in A} \pi_{\theta_t}(s_t, a_t) R_t, \tag{10}$$

where $d^{\pi_{\theta_t}}(s_t)$ is the stationary distribution of MDPs modeling cloud resource allocation under the current policy π_{θ_t} of job scheduling.

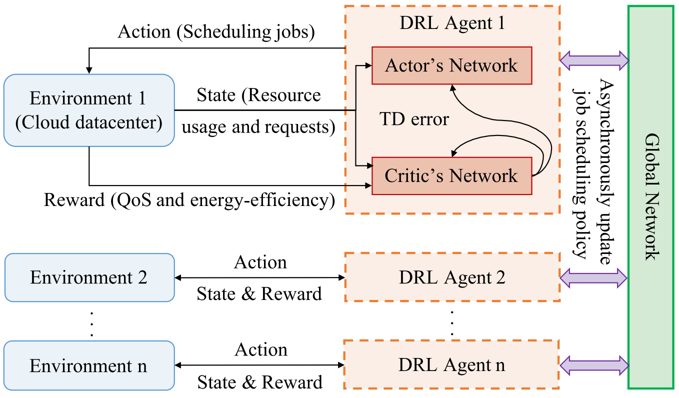


Fig. 4. Framework of the A3C-based cloud resource allocation method.

After the initialization, the training process for optimizing cloud resource allocation begins. To improve the optimization objective, the policy parameters of job scheduling are updated continuously.

In one-step MDPs, the policy gradient of the objective function is defined as

$$\nabla_{\theta_t} J(\theta_t) = E_{\pi_{\theta_t}}[\nabla_{\theta_t} log_{\pi_{\theta_t}}(s_t, a_t) R_t]. \tag{11}$$

When it comes to multi-step MDPs, the instant reward R_t is replaced by the long-term value $Q^{\pi_{\theta_t}}(s_t, a_t)$, and the policy gradient theorem is defined as

Theorem 1. *Policy Gradient Theorem* [13]: For any differentiable policy $\pi_{\theta_t}(s_t, a_t)$ and any policy objective functions, the corresponding gradient is defined as

$$\nabla_{\theta_t} J(\theta_t) = E_{\pi_{\theta_t}}[\nabla_{\theta_t} log\pi_{\theta_t}(s_t, a_t) Q^{\pi_{\theta_t}}(s_t, a_t)]. \tag{12}$$

Based on this theorem, a Temporal Difference (TD) learning [13] is adopted, which estimates the state-values accurately and guides the update of policy parameters.

Fig. 4 illustrates the framework of the proposed A3C-based cloud resource allocation method. Through taking advantage of both policy-based and value-based DRL, the proposed method is able to handle a large action space and reduce the variance when estimating gradient.

In each DRL agent, the critic's network estimates the state-action value function $Q_w(s_t, a_t) \approx Q^{\pi_{\theta_t}}(s_t, a_t)$ and updates parameter w. Moreover, the actor's network guides the update of policy parameters θ_t of the job scheduling policy π_{θ_t} based on the evaluated values from the critic's network. The corresponding policy gradient is defined as

$$\nabla_{\theta_t} J(\theta_t) = E_{\pi_{\theta_t}}[\nabla_{\theta_t} log\pi_{\theta_t}(s_t, a_t) Q_w(s_t, a_t)]. \tag{13}$$

Next, a state-value function $V^{\pi_{\theta_t}}(s)$ is used to reduce the variance when estimating the gradient, which is only related to the state and does not change the gradient. Thus, the policy gradient is redefined as

$$\nabla_{\theta_t} J(\theta_t) = E_{\pi_{\theta_t}}[\nabla_{\theta_t} log\pi_{\theta_t}(s_t, a_t) A^{\pi_{\theta_t}}(s_t, a_t)], \tag{14}$$

where $A^{\pi_{\theta_t}}(s_t, a_t) = Q^{\pi_{\theta_t}}(s_t, a_t) - V^{\pi_{\theta_t}}(s_t)$ is the advantage function. Moreover, $V^{\pi_{\theta_t}}(s)$ is updated by the TD learning, where the TD error is defined as

$$\delta^{\pi_{\theta_t}} = R_t + \beta V^{\pi_{\theta_t}}(s_{t+1}) - V^{\pi_{\theta_t}}(s_t). \tag{15}$$

To improve the training efficiency, multiple DRL agents work simultaneously and update their policy parameters of job scheduling asynchronously, as shown in Algorithm 2. Specifically, a certain number of DRL agents are initialized with the same local parameters of neural networks (i.e., the scheduling policy) and interact with their corresponding environments of cloud datacenters. For each DRL agent, the gradients are accumulated periodically in the actor's and critic's networks and the asynchronous update is executed for the parameters in the global network by using gradient ascent via RMSProp optimizer [22]. Next, each DRL agent pulls the latest parameters of the actor's and critic's networks from the global network and uses them to replace the local parameters. Based on the updated local parameters, each DRL agent will continue to interact with its corresponding environment and independently optimize its local parameters of scheduling policy. Note that there is no coordination among these DRL agents during the local training process. The A3C-based method will be kept training by using the asynchronous update mechanism among multiple DRL agents until the results converge.

Algorithm 2. The Asynchronous Update of Policy Parameters of Job Scheduling in Each DRL Agent

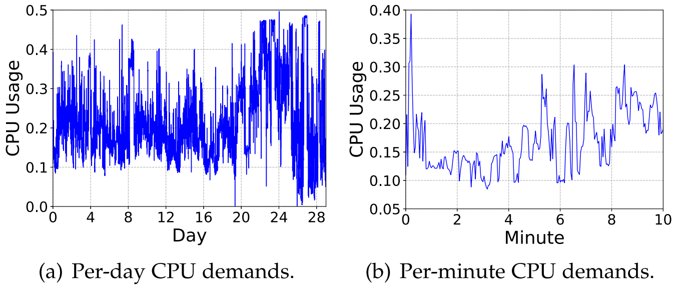
```
1 Initialize: The global and local parameters ($\theta$ and $\theta'$) for
  actor's networks, the global and local parameters ($w$ and $w'$)
  for critic's networks.
2 for $i = temp - u, temp - u + 1, \ldots, temp$ do
3     Accumulate gradients in the actor:
      $d\theta \leftarrow d\theta + \nabla_{\theta'} log\pi_{\theta'}(s_i, a_i)(R_i - V_w(s_i))$;
4     Accumulate gradients in the critic:
      $dw \leftarrow dw + \partial(R_i - V_w(s_i))^2/\partial w'$;
5 end
6 Update global parameters by using gradient ascent via
  RMSProp: $\theta = \theta + \gamma_a d\theta,\ w = w + \gamma_c dw$;
7 Synchronize local parameters: $\theta' = \theta,\ w' = w$;
8 Reset gradients: $d\theta \leftarrow 0,\ dw \leftarrow 0$;
```

5 Performance Evaluation

In this section, we first describe the settings and datasets in our simulation experiments. Next, we evaluate the performance of the proposed method and conduct comparative experiments with other baselines.

5.1 Settings and Datasets

The proposed model of cloud resource allocation is implemented based on the TensorFlow 1.4.0. A cloud datacenter is simulated with 50 heterogeneous servers, where the fraction k of energy consumption for an idle server is set to 70% and the maximum energy consumption P_{max} of a server is set to 250 W [37]. Therefore, the energy consumption of a server is distributed between 175 W and 250 W with the increase of resource usage from 0% to 100%. Moreover, the real-world trace data from Google cloud datacenters [8] is used as the input of our proposed model. The datasets contain the resource usage data of different jobs over 125,000 servers in Google cloud datacenters during May 2011. More specifically, 50 servers are first randomly extracted from Google datasets over 29 days, where each server consists of



(a) Per-day CPU demands. (b) Per-minute CPU demands.

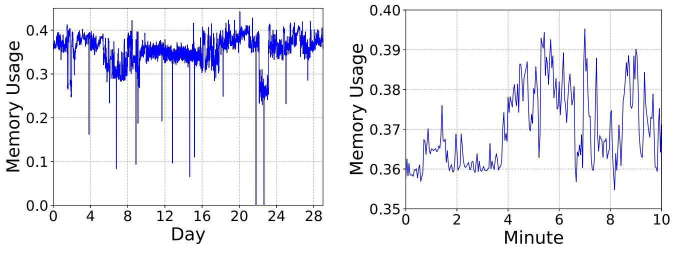
Fig. 5. Varying CPU demands of jobs at different times.

around 100,000 job traces. Next, several essential metrics are extracted from each job trace, including machine ID, job ID, start time, end time, and the corresponding resource usage. For example, Figs. 5 and 6 depict the per-day and per-minute resource (CPU and memory) usage of a server, and they reflect the ever-changing resource demands of jobs at different times. The job duration is assumed to be known before the scheduling. This assumption is reasonable because users commonly specify the requirements of their jobs (including resource usage and job duration) when they want to utilize cloud resources to execute their jobs, which enables cloud datacenters to allocate the resources correspondingly. In addition, the length of job sequence is set to 1000.

During the training process, 10 DRL agents are used to implement the asynchronous update of policy parameters. In each DRL agent, the job trace data is fed in the proposed model by batches, where the batch size is set to 64. As for the design of DNNs, two fully-connected hidden layers are built with 200 and 100 neurons, respectively. Moreover, we set the maximum number of epochs as 1000, the reward decay rate λ as 0.9, and the critic's learning rate γ_c as 0.01. Based on the above settings, extensive simulation experiments are conducted to evaluate the performance of the proposed A3C-based cloud resource allocation method.

To analyze the effectiveness and advantage of the proposed method for cloud resource allocation, extensive comparative experiments are conducted. On one hand, the performance of two advanced DRL-based methods (i.e., PG [20] and DQL [21]) are assessed. On the other hand, five classic algorithms are also evaluated as follows.

- Random. Jobs are executed by a random order of job durations.
- Longest job first (LJF) [25]. Jobs are executed by a decreasing order of job durations.
- Shortest job first (SJF) [25]. Jobs are executed by an increasing order of job durations.



(a) Per-day memory demands. (b) Per-minute memory demands.

Fig. 6. Varying memory demands of jobs at different times.

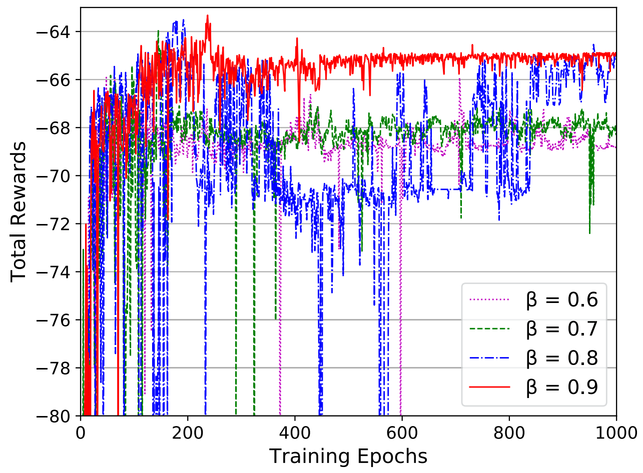


Fig. 7. Convergence versus different TD error discount factors.

- Round-robin (RR)[26]. Jobs are executed fairly in a circular order, where time slices are employed and assigned to each job in equal portions.
- Tetris [27]. Jobs are executed based on their resource demands and the availability of system resources at the moment they arrive.

5.2 Convergence Results

To evaluate the convergence of the proposed A3C-based cloud resource allocation method, the impact of two essential parameters is investigated, including the TD error discount factor β and the actor's learning rate γ_a.

First of all, the value of TD error discount factor β is changed with the constant actor's learning rate $\gamma_a = 0.001$. As shown in Fig. 7, higher total rewards and faster convergence (around 400 training epochs) are achieved when β is set to 0.9. This is because the proposed method with $\beta = 0.9$ makes better use of recent rewards to guide the actor's network and thus better actions are chosen along with the right direction for higher total rewards. Therefore, $\beta = 0.9$ will be used in the following experiments.

Next, the constant TD error discount factor $\beta = 0.9$ is used to analyze the convergence of our proposed method with the different values of actor's learning rate γ_a. As shown in Fig. 8, when γ_a is set to a large value (e.g., 0.1 or 0.01), high total rewards can be obtained in few training epochs. However, the algorithm converges to the local optimum in this case, and thus it can no longer learn a more optimized policy. By contrast, when γ_a is set to 0.001, it only takes around 400 training epochs to achieve higher total rewards than the above two cases. When γ_a decreases to 0.0001, the learning curve always fluctuates strongly with the increase of training epochs, and it is hard to reach a smooth convergence in this case. Thus, $\gamma_a = 0.001$ is more suitable for the next experiments than other values.

5.3 Comparison Under Single-Objective Optimization

In this subsection, the proposed A3C-based cloud resource allocation method is first evaluated by several performance metrics, including the total rewards, QoS (normalized average job latency and job dismissing rate), and energy-efficiency

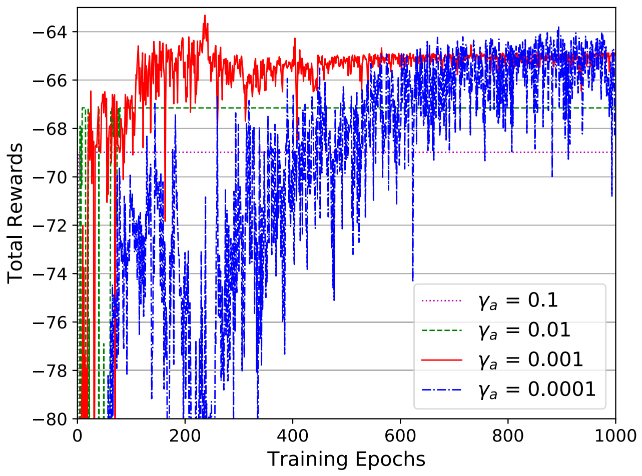


Fig. 8. Convergence versus different actor's learning rates.

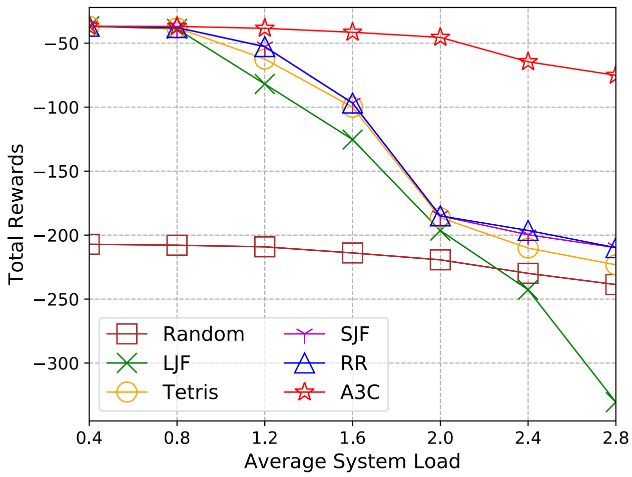


Fig. 9. Total rewards of different resource allocation methods with various system loads under single-objective optimization.

(average energy consumption of jobs), under different cases with various average system loads. Next, the proposed method is compared with some classic resource allocation methods, including LJF, Tetris, SJF, and RR, under single-objective (QoS) optimization.

As shown in Fig. 9, the total rewards (represent the QoS) generally declines with the increase of average system load. In this case, the proposed method can always achieve higher total rewards than other methods even if the average system load becomes higher. By contrast, other classic methods present comparable performance only when the average system load is less than 1.2. Especially, when the average system load is over 2.0, the performance of these classic methods is only slightly better than the random scheme. The LJF method even performs worse than the random scheme when the average system load is over 2.4. This is because that a large number of jobs are waiting to be processed when the average system load is high but the LJF method always schedules the job with the longest job duration in priority, which results in the excessive waiting of many jobs and seriously degrades the scheduling performance. By contrast, the proposed method always maintains excellent performance. The results verify the advantage of the proposed method in scheduling jobs under complicated environments with high system loads.

As shown in Figs. 10a and 10c, the proposed method obtains both the lowest normalized average job latency and job dismissing rate among all these methods. Especially, the performance gap becomes larger with the increase of average system load. This also verifies the strong adaptiveness of our proposed method in dynamic cloud environments with changeable average system loads. Besides, the average job energy consumption is also measured in this case, and the comparisons are conducted among these methods under the case of single-objective optimization. As shown in Fig. 10b, the proposed method leads to the higher average energy consumption of jobs when the average system load stays low (i.e., less than 2), although more energy consumption can be reduced when the average system load is over 2.

5.4 Comparison Under Multi-Objective Optimization

In this subsection, the comparative experiments are conducted between the proposed method and other classic methods for cloud resource allocation under multi-objective (QoS and energy-efficiency) optimization. As shown in Fig. 11, the total rewards (represent the weighted sum of the QoS and energy-efficiency) degrade with the increase of average system load. This is because the growing and changeable demands from user jobs increase the complexity of cloud resource allocation. In this case, the proposed method obtains much higher total rewards than other classic resource allocation methods when the average system load is high. Especially, when the average system load is over 1.6 with more complicated system states, the performance improvement

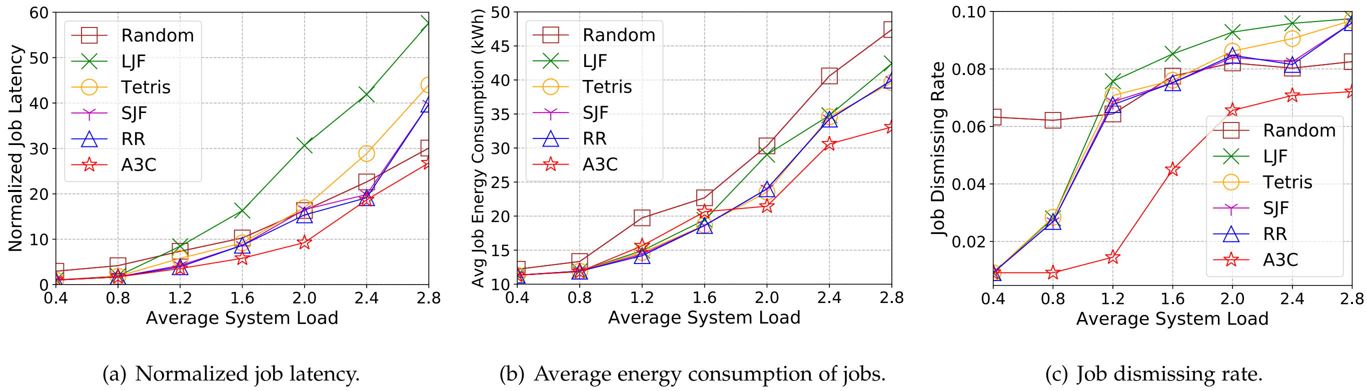


Fig. 10. Comparison of performance metrics among different resource allocation methods under single-objective optimization.

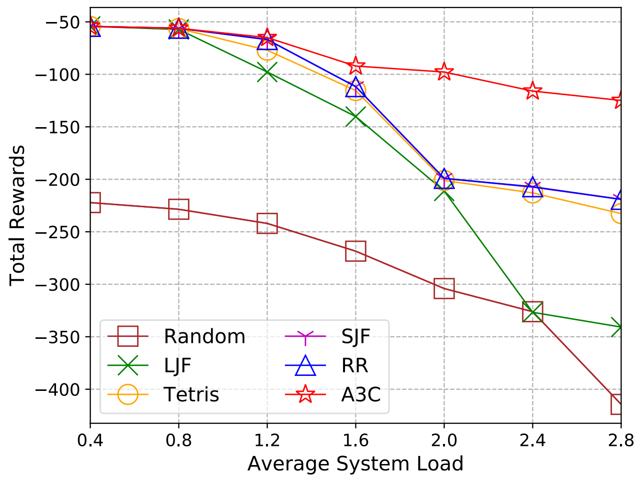


Fig. 11. Total rewards of different resource allocation methods with various system loads under multi-objective optimization.

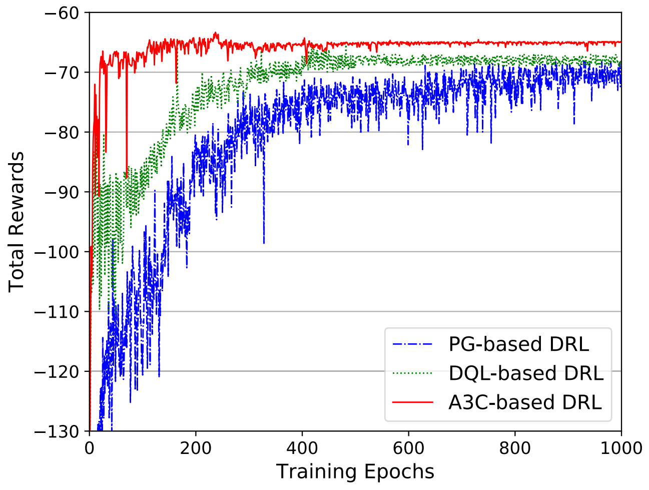


Fig. 13. Total rewards of different DRL-based methods under multi-objective optimization when the average system load = 1.2.

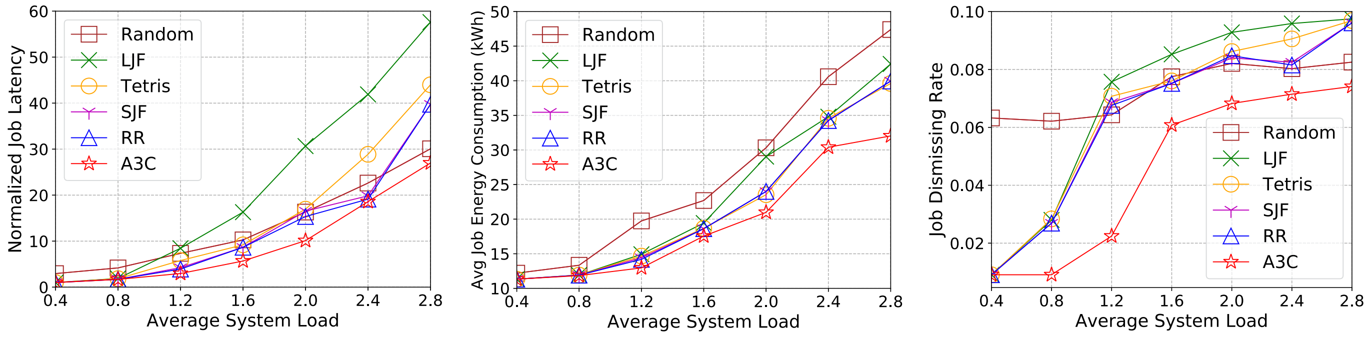
achieved by the proposed method becomes more obvious. This is because the proposed method is of good ability to find a better trade-off between the QoS and energy-efficiency during the job scheduling process.

As shown in Figs. 12a and 12c, the proposed method outperforms other resource allocation methods in terms of both normalized average job latency and job dismissing rate. This verifies that the proposed method has the excellent stability of maintaining the superior QoS in the cases of both single-objective and multi-objective optimizations. Moreover, Fig. 12b depicts the energy-efficiency of different resource allocation methods, where the proposed method can always attain the lowest average energy consumption of jobs among these methods with the increase of average system load. Thus, the proposed method makes up for the defects that occur in Fig. 10b under the case of single-objective optimization. This is because that the energy-efficiency is integrated in our DRL-based scheduling method, and thus both the QoS and energy-efficiency are well considered during the job scheduling. The above results demonstrate the advantageous performance of our proposed method in improving the QoS and energy-efficiency.

5.5 Comparison Among Different DRL-Based Methods

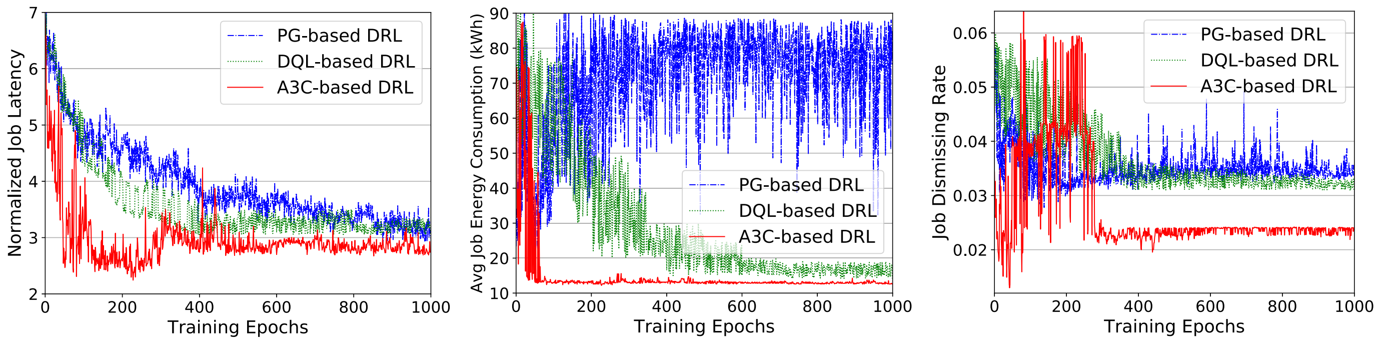
In this subsection, the performance comparison between the proposed A3C-based method and two advanced DRL-based methods for cloud resource allocation is conducted under multi-objective optimization, where the average system load is 1.2. As shown in Fig. 13, the proposed method can always achieve higher total rewards than the other two DRL-based methods during the training process of resource optimization. Moreover, the learning curve of the proposed method tends to converge after around 200 training epochs. However, the PG-based and DQL-based methods respectively require around 800 and 400 training epochs to reach a relatively-smooth convergence. As shown from Figs. 14a, 14b, 14c, and 14d, the proposed method can achieve the better QoS (normalized average job latency and job dismissing rate) and higher energy-efficiency (average energy consumption of jobs) compared to the other two DRL-based methods. Therefore, the above results demonstrate the excellent performance and high training efficiency of the proposed method. This is because the proposed method is able to effectively avoid large variance by using the advantage function when estimating the policy gradient. Meanwhile, the efficient convergence can be achieved by using the asynchronous update mechanism among different DRL agents.

Finally, the detailed performance metrics of the proposed A3C-based method and other classic methods for cloud resource allocation are exhibited when the average system load is 1.2. As shown in Table 2, the proposed method reduces over 3% normalized average job latency and 27%



(a) Normalized job latency. (b) Average energy consumption of jobs. (c) Job dismissing rate.

Fig. 12. Comparison of performance metrics among different resource allocation methods under multi-objective optimization.



(a) Normalized job latency. (b) Average energy consumption of jobs. (c) Job dismissing rate.

Fig. 14. Comparison of performance metrics among different DRL-based methods.

job dismissing rate than the DQL-based DRL method, which performs best among other methods in terms of these two metrics. The PG-based DRL method results in the worst energy-efficiency (average energy consumption of jobs) among all these methods. This is because the PG-based DRL generates high variance when estimating the policy gradient. Therefore, it cannot achieve a good load balancing among different servers. Consequently, it will result in high loads and low utilization on servers, causing excessive energy consumption. By contrast, around 9% of the average energy consumption is saved by using the proposed method compared to the RR method, which leads to the lowest average energy consumption of jobs among other methods. Moreover, the proposed method can achieve excellent QoS and energy-efficiency simultaneously. This is because that the training efficiency of the proposed method is greatly improved by using the asynchronous update of policy parameters among multiple DRL agents. Therefore, the proposed method can efficiently approach the global optimal and well guarantee the QoS and energy-efficiency simultaneously.

6 Conclusion

In this paper, we first formulate the resource allocation issue in cloud datacenters as a model-free DRL problem with dynamic system states and various user demands. Next, we propose an A3C-based resource allocation method to effectively schedule jobs for improving the QoS and energy-efficiency in cloud datacenters. The extensive simulation experiments using real-world trace data from Google cloud datacenters demonstrate the effectiveness of the proposed method in achieving adaptive and efficient resource allocation. More specifically, the proposed method outperforms the classic resource allocation methods (i.e., LJF, Tetris, SJF, RR, PG, and DQL) in terms of the QoS (normalized average job latency and job dismissing rate) and energy-efficiency (average energy consumption of jobs). Moreover, the proposed method works better than the others with the increase of average system load, and it can achieve higher training efficiency (faster convergence) than two advanced DRL-based methods (i.e., PG and DQL). The simulation results show the great value of the proposed method for improving resource allocation in cloud datacenters.

In our future work, we plan to first extend the proposed model to consider the jobs' priority order. Specifically, we will need to redefine the state space, action space, and reward function, taking into account the jobs' priority order. For example, the actions of scheduling the high-priority jobs can lead to better rewards, thus guiding the algorithm to learn scheduling policies considering the jobs' priority order. Next, we intend to design a new query-aware database parameter tuning method using an advanced DRL model built on this work. Through feeding the features of query information, the DRL model could learn the relations among database states, queries, and configurations to realize the automatic parameter tuning. Moreover, we will try to improve the generalization of the proposed DRL-based resource allocation scheme by developing an automatic data augmentation technique, which aims to regularize policies and value functions with respect to various state transitions and thus allows the DRL agent to capture task invariances and learn useful behaviors when the environment changes.

TABLE 2
Performance Metrics (Normalize Average Job Latency, Average Energy Consumption of Jobs, and Job Dismissing Rate) With Average Load = 1.2

Methods	Performance metrics		
	Latency	*Energy (kWh)*	*Dismissing Rate*
Random	7.3448	19.7431	0.0643
LJF	8.4531	14.9273	0.0757
Tetris	5.7844	14.6598	0.0707
SJF	4.2325	14.4228	0.0686
RR	3.9226	14.1897	0.0676
PG-based DRL	3.1296	74.6051	0.0335
DQL-based DRL	3.0583	14.2016	0.0310
A3C-based DRL	2.9659	12.9542	0.0224

References

[1] Z. Chen, J. Hu, and G. Min, "Learning-based resource allocation in cloud data center using advantage actor-critic," in *Proc. 53rd Int. Conf. Commun.*, 2019, pp. 1–6.
[2] B. Wan, J. Dang, Z. Li, H. Gong, F. Zhang, and S. Oh, "Modeling analysis and cost-performance ratio optimization of virtual machine scheduling in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1518–1532, Jul. 2020.
[3] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 909–922, Apr. 2020.

[4] Z. Chen, J. Hu, G. Min, A. Zomaya, and T. El-Ghazawi, "Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 923–934, Apr. 2020.

[5] X. Liu, W. Li, and X. Zhang, "Strategy-proof mechanism for provisioning and allocation virtual machines in heterogeneous clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1650–1663, Jul. 2018.

[6] Y. Zhan, S. M. Ghamkhari, H. Akhavan-Hejazi, D. Xu, and H. Mohsenian-Rad, "Cost-aware traffic management under demand uncertainty from a colocation data center user's perspective," *IEEE Trans. Serv. Comput.*, vol. 14, no. 2, pp. 400–412, Mar./Apr. 2021.

[7] S. Hsieh, C. Liu, R. Buyya, and A. Y. Zomaya, "Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers," *J. Parallel Distrib. Comput.*, vol. 139, pp. 99–109, 2020.

[8] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + schema," *Menlo Park, CA, USA, Google White Paper*, 2011. Accessed: Nov. 12, 2018. [Online]. Available: https://github.com/google/cluster-data/blob/master/bibliography.bib.

[9] F. Tseng, X. Wang, L. Chou, H. Chao, and V. C. Leung, "Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1688–1699, Jun. 2018.

[10] F. Zahid, A. Taherkordi, E. G. Gran, T. Skeie, and B. D. Johnsen, "A self-adaptive network for HPC clouds: Architecture, framework, and implementation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2658–2671, Dec. 2018.

[11] S. Khatua, P. K. Sur, R. K. Das, and N. Mukherjee, "Heuristic-based resource reservation strategies for public cloud," *IEEE Trans. Cloud Comput.*, vol. 4, no. 4, pp. 392–401, Oct.–Dec. 2016.

[12] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Trans. Internet Technol.*, vol. 19, no. 2, 2019, Art. no. 23.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[14] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. Lau, "Dynamic virtual machine management via approximate markov decision process," in *Proc. 35th Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[15] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[17] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep q-learning," *Inf. Sci.*, vol. 512, pp. 1170–1191, 2020.

[18] D. Yi, X. Zhou, Y. Wen, and R. Tan, "Toward efficient compute-intensive job allocation for green data centers: A deep reinforcement learning approach," in *Proc. 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 634–644.

[19] N. Liu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 372–382.

[20] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th Workshop Hot Top. Netw.*, 2016, pp. 50–56.

[21] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[22] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[23] A. Brandwajn, T. Begin, H. Castel-Taleb, and T. Atmaca, "A study of systems with multiple operating levels, probabilistic thresholds and hysteresis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 748–757, Apr. 2018.

[24] Y. Xiong, S. Huang, M. Wu, J. She, and K. Jiang, "A johnson's-rule-based genetic algorithm for two-stage-task scheduling problem in data-centers of cloud computing," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 597–610, Jul.–Sep. 2019.

[25] J. Wang, D. Han, and R. Wang, "A new rule-based power-aware job scheduler for supercomputers," *J. Supercomput.*, vol. 74, no. 6, pp. 2508–2527, 2018.

[26] P. Samal and P. Mishra, "Analysis of variants in round robin algorithms for load balancing in cloud computing," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 3, pp. 416–419, 2013.

[27] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. 36th Int. Conf. Appl., Technol., Architectures, Protoc. Comput. Commun.*, 2014, pp. 455–466.

[28] P. Haratian, F. Safi-Esfahani, L. Salimian, and A. Nabiollahi, "An adaptive and fuzzy resource management approach in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 907–920, Oct.–Dec. 2019.

[29] M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu, "Feedback autonomic provisioning for guaranteeing performance in MapReduce systems," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1004–1016, Oct.–Dec. 2018.

[30] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[31] Y. Zhang, J. Yao, and H. Guan, "Intelligent cloud resource management with deep reinforcement learning," *IEEE Cloud Comput.*, vol. 4, no. 6, pp. 60–69, Nov./Dec. 2018.

[32] M. Eisen, C. Zhang, L. F. Chamon, D. D. Lee, and A. Ribeiro, "Learning optimal resource allocations in wireless systems," *IEEE Trans. Signal Process.*, vol. 67, no. 10, pp. 2775–2790, May 2019.

[33] Z. Fang, T. Yu, O. J. Mengshoel, and R. K. Gupta, "QoS-aware scheduling of heterogeneous servers for inference in deep neural networks," in *Proc. 26th Int. Conf. Inf. Knowl. Manage.*, 2017, pp. 2067–2070.

[34] Y. Wei, F. R. Yu, M. Song, and Z. Han, "User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 17, no. 1, pp. 680–692, Jan. 2018.

[35] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "GA3C: GPU-based A3C for deep reinforcement learning," in *Proc. 30th Conf. Neural Inf. Process. Syst. Workshop*, 2016, pp. 1–6.

[36] M. T. Islam, S. N. Srirama, S. Karunasekera, and R. Buyya, "Cost-efficient dynamic scheduling of big data applications in apache spark on cloud," *J. Syst. Softw.*, vol. 162, pp. 1–14, 2020.

[37] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[38] X. Zhou, K. Wang, W. Jia, and M. Guo, "Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers," in *Proc. 25th Int. Symp. Qual. Serv.*, 2017, pp. 1–6.

[39] J. Mei, K. Li, A. Ouyang, and K. Li, "A profit maximization scheme with guaranteed quality of service in cloud computing," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3064–3078, Nov. 2015.

Zheyi Chen received the BSc degree in computer science from Shanxi University, China, in 2014 and the MSc degree in computer science from Tsinghua University, China, in 2017. He is currently working toward the PhD degree in computer science with the University of Exeter, U.K. His research interests include cloud computing, mobile edge computing, deep learning, reinforcement learning, and resource optimization.

Jia Hu received the BEng and MEng degrees in electronic engineering from the Huazhong University of Science and Technology, China, in 2004 and 2006, respectively, and the PhD degree in computer science from the University of Bradford. He is currently a senior lecturer in computer science with the University of Exeter. His research interests include edge-cloud computing, resource optimization, applied machine learning, and network security.

Geyong Min received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995 and the PhD degree in computing science from the University of Glasgow, U.K., in 2003. He is currently a professor of high-performance computing and networking with the Department of Computer Science, University of Exeter, U.K. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, modeling and performance engineering.

Chunbo Luo received the BEng degree in electronic information engineering from the University of Electronic Science and Technology of China in 2005 and the PhD degree in high-performance cooperative wireless networks from the University of Reading, U.K, in 2011. He is currently a senior lecturer in computer science with the University of Exeter, UK. His research interests include developing model-based and machine learning algorithms to solve networking and engineering problems, including wireless networks, with a particular focus on unmanned aerial vehicles.

Tarek El-Ghazawi (Fellow, IEEE) received the PhD degree in electrical and computer engineering from New Mexico State University in 1988. He is currently a professor with the Department of Electrical and Computer Engineering, The George Washington University, where he leads the university-wide Strategic Academic Program in high-performance computing. He is the founding director of The GW Institute for Massively Parallel Applications and Computing Technologies and the founding co-director of the NSF Industry or University Center for high-performance reconfigurable computing. He has authored or coauthored more than 200 refereed research papers and book chapters in the areas of his research interests, which include high-performance computing, parallel computer architectures, high-performance I or O, reconfigurable computing, experimental performance evaluations, computer vision, and remote sensing. His research has been supported by DoD/DARPA, NASA, NSF, and also industry, including IBM and SGI.