YU ZHANG

JIANGUO YAO

HAIBING GUAN

Intelligent Cloud Resource Management with Deep Reinforcement Learning

Yu Zhang, Jianguo Yao, and Haibing Guan, Shanghai Jiao Tong University

The cloud provides low-cost and flexible IT resources (hardware and software) across the Internet. As more cloud providers seek to drive greater business outcomes and the environments of the cloud become more complicated, it is evident that the era of the intelligent cloud has arrived. The intelligent cloud

faces several challenges, including optimizing the economic cloud service configuration and adaptively allocating resources. In particular, there is a growing trend toward using machine learning to improve the intelligence of cloud management. This article discusses an architecture of intelligent cloud resource management with deep reinforcement learning. The deep reinforcement learning makes clouds automatically and efficiently negotiate the most appropriate configuration, directly from complicated cloud environments. Finally, we give an example to evaluate and conclude the remarkable ability of the intelligent cloud with deep reinforcement learning.

Cloud computing, unlocked by virtualization, debuts as an important service-oriented computing pattern. With the increasingly developed infrastructures and technologies, cloud computing shows its excellent ability in scalability, flexibility, and accessibility. However, due to its low-cost and on-demand availability, it is common for cloud asset misuse to arise, reducing resource utilization or even putting infrastructures at risk. As a result, resource management is the key to provide continuous availability and efficient utilization.

In general, there exist several issues concerning traditional resource management. First, resource management includes the coordination of cloud environment, physical resources, virtual resources, which is a vast project.[1] Besides, resource management is an error-prone process. Traditionally, a large number of cloud systems are configured manually and heavily rely on relevant knowledge and experiences.[2] Despite its inconvenience, inefficiency and high cost are also incurred as results of manual provision, and uncertainties of resource demands for clients increase complexity of the whole system. The last problem is related to resource balance.[3] Along with a large scope of applications, cloud providers need to balance resources to make sure of the priority and timely basis of critical workflows, such as transaction processing. Therefore, there is a strong demand on intelligent cloud resource management.

To deal with these problems, intelligent cloud resource management is introduced as a significant shift to automatically manage and coordinate virtually all aspects of cloud assets, especially resource utilization. Table 1 exhibits several of the most popular intelligent cloud solutions, and it is obvious that intelligent cloud resource management has become a state-of-the-art fashion in cloud computing.

One of the most beneficial aspects of intelligent cloud resource management is online provision, based on which clients can automatically scale resources in an on-demand fashion. To be specific, clients can automatically obtain more resources when workloads are heavy and then release excess resources immediately after, just like a smart thermostat. Another crucial motivation to develop intelligent cloud resource management is the error-prone nature caused by manual configuration;[4,5] therefore, it is important to allocate resources automatically in practice. Compared with classical combinational optimization algorithms[6] negotiating a configuration in a static environment, the approaches using feedback control are able to allocate resources dynamically,[7] and one of the most famous algorithms is deep reinforcement learning (DRL). Due to its excellent ability of autonomic and efficient learning, DRL not only handles the uncertainties of client demands to provide online services but also deals with complex and dynamic cloud environments, and we will elaborate in the next section accordingly.

TABLE 1. Intelligent cloud resource management solutions.

NAME	MAIN FEATURE	FOCUS
Microsoft's intelligent cloud	Secure solution with ability to listen, learn, and predict with artificial intelligence	Flexible infrastructure that can scale on demand Cloud availability and cloud-based infrastructure reliability
Intelligent cloud computing platform of Akamai	Cloud-based platform as foundation for content delivery network and cloud security services	High levels of availability and on-demand scale Around-the-clock insight and control
Apttus Intelligent Cloud platform	Flexible and scalable end-to-end software as a service solution to maximize entire revenue operation	Recommendation of intelligent transaction actions with machine learning Effectiveness across sales and revenue cycle
Zoolz Intelligent cloud storage	Data management platform combining conventional data management tools with artificial intelligence process	Accurate discovery, fast-track accessibility, and efficient organization of files with artificial intelligence Adequate protection of data with military-grade encryption
Intelligent Cloud Resources Inc	Cloud enabler and cloud broker services to small- and medium-sized companies	Efficient cloud migration and cloud management Powerful application development and cloud enablement

In summary, cloud computing brings greater capabilities and opportunities for recent IT resource utilization. As the core of cloud computing, intelligent cloud resource management shows its remarkable ability in online and automatic resource allocation, and it is possible to maximize the benefits of intelligent resource management with efficient control algorithms.

DRL

Reinforcement learning is an area of machine learning, which enables the agent to determine the ideal behavior from its own experience. The most famous algorithm in reinforcement learning is Q-learning, and the main idea of the algorithm is to select actions based on a Q-table. Taking the famous video game *Breakout* as an example, you control a movable paddle at the bottom of the screen and need to bounce the ball to hit the bricks lined on the top of the screen. Each time a brick is hit, it is destroyed, and your score is increased—you get a reward. In Q-learning, the agent (the artificial game player) is input to only several screen images, and it is able to choose a moving action based on its historical experiences, so as to maximize the final score. Generally, historical experiences are stored in a Q-table, together with Q-values, which are used to assess long-term effects of a certain action in a given state. The agent will explore uncharted territory sometimes to avoid locally optimal solutions, and Q-table will update iteratively based on the Bellman equation[8] each time a new action is taken.

In 2006, deep learning was established and has been further developed as a powerful set of techniques in a learning neural network; in detail, it is a set of computational models that include multiple processing layers to learn features of data with multiple levels of abstraction.[9] With the combination of enough such transformations, deep learning has produced extremely promising results in discovering intricate structures of high-dimensional data, such as images, speech, and space data. As derived from the neural network, deep learning is a model-free function approximator and can easily take advantage of the increasing amount of computation and data. On the basis of its remarkable ability in feature extraction and function approximation, DRL is proposed by connecting traditional reinforcement learning to deep learning.

In 2015, Google proposed a novel algorithm called deep Q-network (DQN) that utilizes a Q-network derived from convolutional neural network (CNN) to approximate Q-function in Q-learning, and it is capable of human-level performance on many Atari video games by using unprocessed pixels for inputs.[10] Generally, it is known that there exist some stability issues of reinforcement learning with a nonlinear approximator (e.g., neural network),[11] mainly caused by the correlated inputs, potentially oscillating policies, and unstable Q-learning gradients. Fortunately, on the basis of the experience replay, Q-network freeze, and reward normalization, DQN addresses these problems and dramatically improves state-of-the-art technologies in speech recognition,

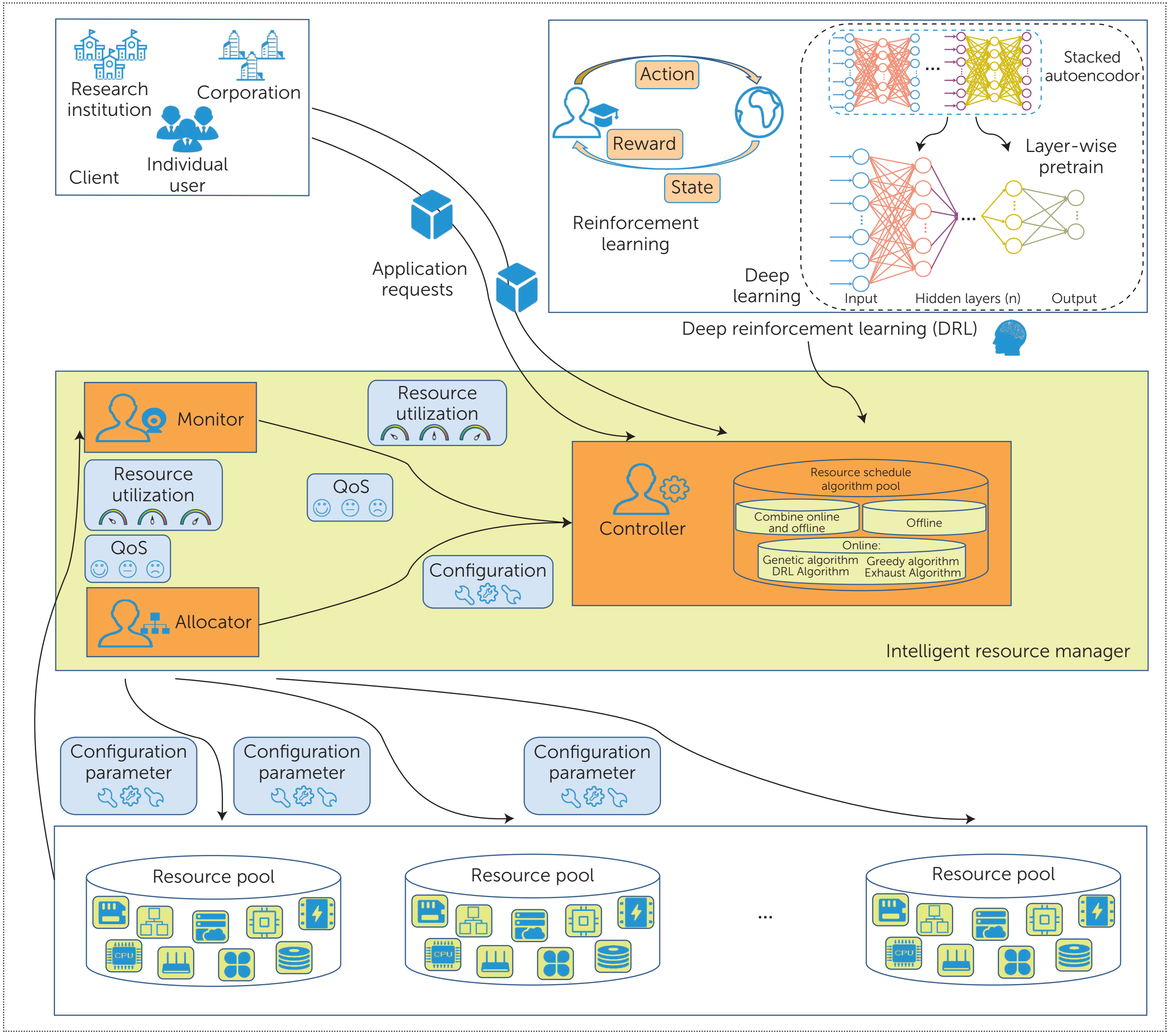

FIGURE 1. Architecture of intelligent resource management by using deep reinforcement learning.

visual object detection, and many other domains, such as drug discovery and genomics.

When Clouds Meet DRL: An Intelligent Resource Management Architecture

As Figure 1 shows, an intelligent resource management architecture, mainly contains two components: a intelligent resource manager, which is composed of *controller*, *monitor*, and *allocator* and an IT resource, which consists of extensive resource pools.[12] Clients first communicate with the controller to submit application requests with various demands. Based on application demands and current resource utilization information, the controller implements the algorithm chosen from its resource schedule algorithm pool to meet application demands, while respecting system resource constraint. The resource schedule algorithm pool, which plays an important role in intelligent resource management architecture, includes different kinds of algorithms, such as offline and online algorithms and algorithms combining both online and offline parts. The monitor is responsible for gathering information of system resource utilization and application quality of service (QoS) to update the controller periodically, and the allocator is in charge of mapping applications to resource pools according to the configuration negotiated by the controller.

The controller is the key part of a resource management architecture, as it not only figures out the (near-) optimal configuration policy but also coordinates with the monitor and allocator to allocate resources intelligently. The heart of the controller is a resource schedule algorithm pool, which contains plenty of control algorithms. The DRL algorithm presented in this paper is an online algorithm, which connects reinforcement learning with deep learning to generate the (near-) optimal resource configuration in limited iterations directly from raw application demands, especially for high-dimensional demands. As Figure 1 shows, the controller selects an action according to the deep neural network and then obtains feedback information as a reward and a new environment state from the application operating environment. The deep neural network is pretrained through the stacked autoencoder (SA), followed by using reinforcement learning experiences for optimization. Therefore, the reinforcement learning and deep learning part can fully cooperate to process raw application demands and figure out a configuration policy intelligently in finite steps.

The resource pool is a fully managed cloud hosting solution with excellent flexibility. For server providers, a resource pool represents a set of strategies to categorize and manage their resources, and for users, it is an abstraction used to present and consume resources in a consistent fashion. In general, a resource pool contains five layers: physical resource, virtual resource, hypervisor, virtual machine, and application. The allocator maps applications to the corresponding resource pools and then allocates appropriate resources for implementation.

In conclusion, the controller, monitor, and allocator coordinate with each other to intelligently allocate resources, while respecting two constraints: one is that QoS requirements of applications must be met, and the other is the amount of resource usage must be less than the total amount of available resources in the system.

An Example Study

In DRL algorithms, such as DQN, the most common method to formalize a reinforcement learning problem is to represent it as a Markov decision process (MDP). Suppose the agent is situated in an environment, which is represented by a certain state s. The agent can perform certain actions in the environment, then receives the reward, and transforms the environment to a new state. The experience of a transition is expressed as s,a,r,s', and sets of states, actions, together with transition rules, make up an MDP. One episode of this process forms a finite sequence of states, actions and rewards:

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots s_t, a_t, r_t, s_{t+1}, \ldots, s_{n-1}, a_{n-1}, r_{n-1}, s_n, \quad (1)$$

where s_t denotes the current state, a_t denotes the current action, r_t denotes the reward after performing an action, and s_{t+1} denotes the next state.

In resource management, given sparse but complex relationships among various resources, we propose a modified DQN algorithm called SA Q-network (SAQN), combining the SA with Q-learning, which enables the agent to negotiate the (near-) optimal resource configuration policy directly from raw application inputs, especially high-dimensional data.[13] The target of resource management is related to numerous measurement indexes, such as performance, cost, efficiency, and durability. To clearly elaborate the SAQN algorithm, here we set the target to find the optimal configuration policy, respecting performance constraints with the minimal cost.

On the basis of the predefined target, in SAQN, a state refers to a configuration of multiple resources, such as the amounts of CPUs and memories. An action refers to one-unit adjustment of a single resource, such as one-unit increase of CPU. A reward refers to a two-dimensional vector involving performance and cost, and we abandon the method of representing a reward as a linear combination of performance and cost, due to its complexity and requirements for experience in parameter selection. Other adjustable parameters generally start from a common fixed value and then are adjusted based on system performance, such as the discount factor starts from 0.5, which enables the agent to balance the current and future estimation of rewards. Because a new state depends only on the current state and action, which satisfies the Markov assumption, the process of negotiating the optimal configuration can be formalized as an MDP as in DQN. The agent can then choose an action through its selection policy and transform to a new state according to the transition rule.

After storing the finished transition sequence into a replay memory, it is the point in which deep learning steps in. Different from the Bellman equation that updates and estimates the Q-function separately, SAQN chooses a nonlinear approximator (SA network) to estimate the Q-function, which takes states and actions as inputs and outputs the corresponding Q-values. As the Q-value is used to assess long-term effects of a certain action in a given state and the purpose of SAQN is to learn the optimal

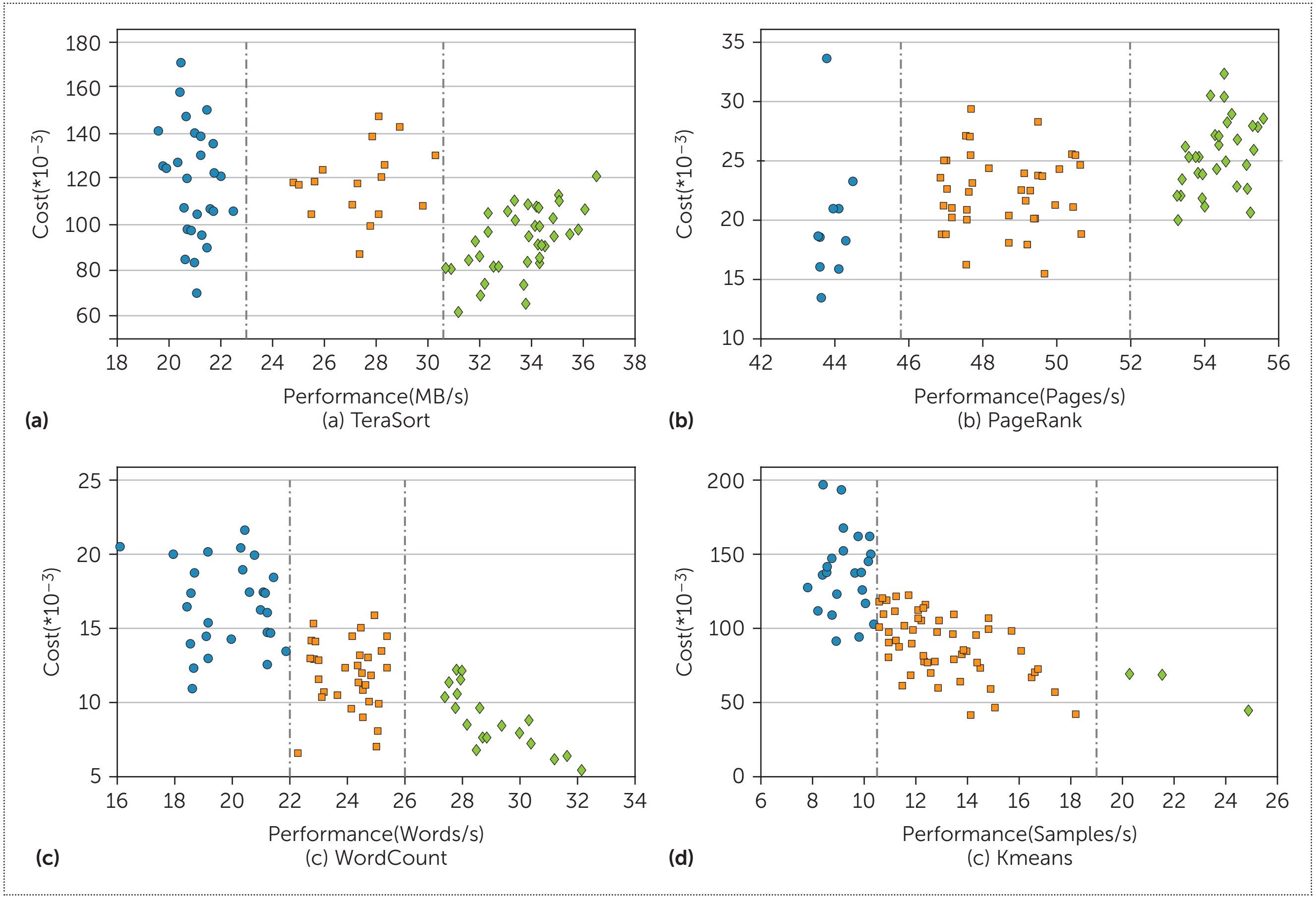

FIGURE 2. The measurements of performance and cost for TeraSort, PageRank, WordCount, and k-means.

configuration policy that satisfies the predefined performance with the minimal cost, the Q-value in SAQN is a two-dimensional vector consisting of performance and cost, which is exactly the normalized reward. In SAQN, the loss function is defined as

$$L = \left[r_t + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}\right) - Q(s_t, a_t) \right]^2, \qquad (2)$$

where r_t is the reward observed for the current state s_t and action a_t, s_{t+1} is the next state, and γ is the discount factor ($\gamma \in (0,1)$), which trades off the importance of sooner versus later rewards. Like other neural networks, the optimal configuration can be reached by stochastic gradient descent.

However, there are stability issues raised by using a nonlinear function approximator in a reinforcement learning algorithm. First, as the input data is sequential, it does not follow the independent identical distribution required by neural networks. To tackle this problem, we adopt a mechanism called experience replay that stores experienced transition sequences in a replay memory so as to train Q-network. As a result, it not only breaks the correlation in data but also enables the network to learn from all past experiences. Moreover, the policy may oscillate with slight changes to Q-values, and the distribution of data will swing largely. The common way to address this problem is to freeze Q-network and update fixed network parameters periodically.[14] The last problem is the unknown scale of rewards, which will lead naive Q-learning gradients to an unstable state when backpropagated. To solve this, we clip rewards and normalize the network adaptively to a sensible range, which also limits error derivations.

The whole execution flow of SAQN can be described as follows. First, the controller communicates with the agent about client application demands. The agent then starts from the initial configuration, explores the environment, and exploits its current knowledge to choose an action. Here, we adopt an adaptive ε-greedy exploration policy, which sets the agent to randomly explore all possible actions at the beginning and then to more rely on its experience. On the basis of the selected action

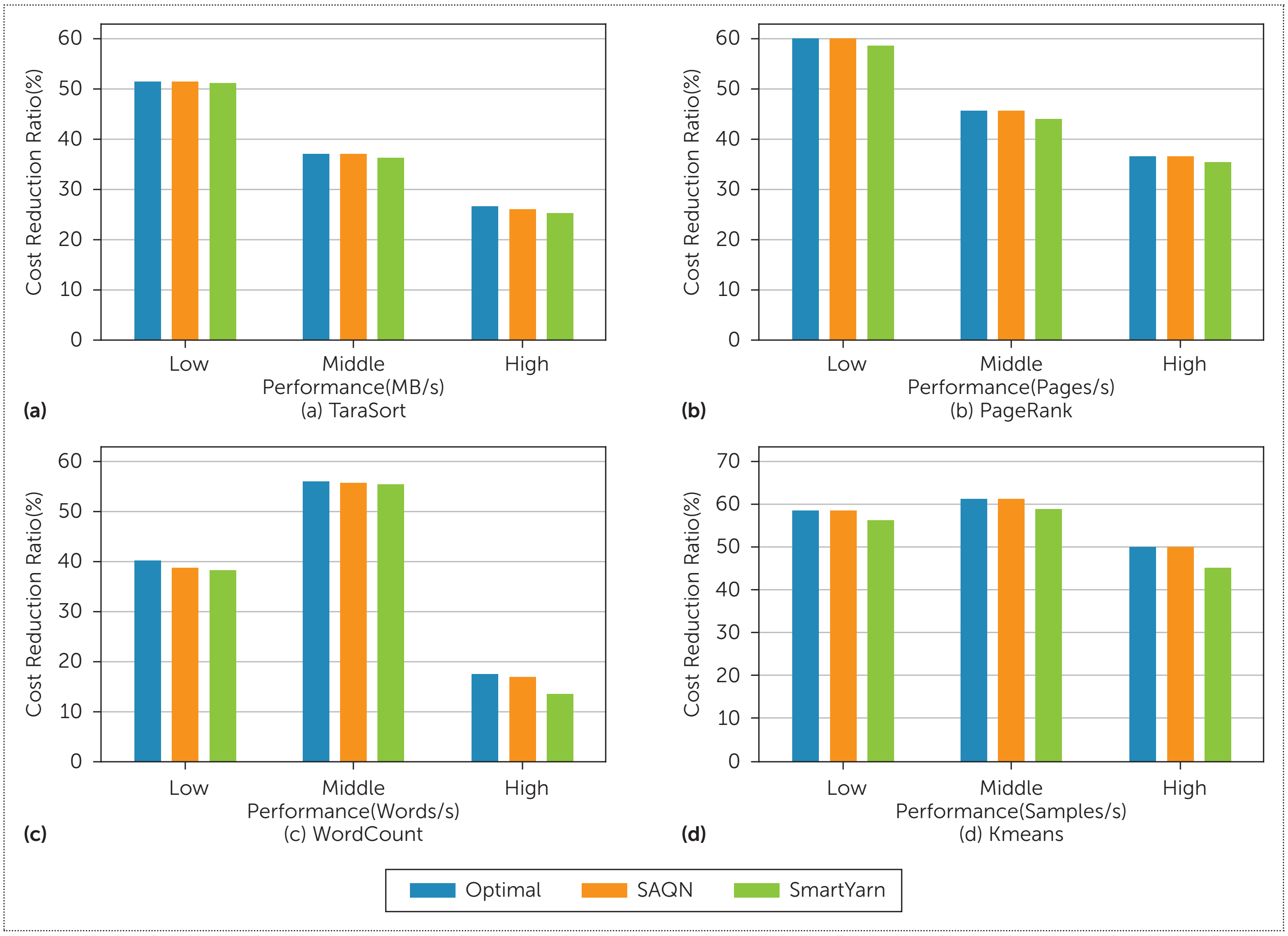

FIGURE 3. Accuracy comparisons among the optimal configuration, Smart YARN, and stacked autoencoder Q-network (SAQN).

and transition rule, the agent receives a feedback reward and steps into a new state. After finishing a transition sequence, the agent stores it into a replay memory to train Q-network. During the network training stage, the agent samples random transition sequences from the replay memory and then trains the Q-network on the basis of the loss function. When the training is done, the agent steps into the next iteration on the basis of the new state until the optimization goal is reached.

To evaluate the performance of SAQN, we design experiments with 4 IntelR Xeon E5645 servers and HiBench in Apache Hadoop (YARN) 2.6.0.[15] The benchmarks we choose are TeraSort, WordCount, PageRank, and *k*-means, which are typical in Hibench. There are four types of resources, namely, CPU and memory in the map and reduce stage, the total available number of each resource scales from 1 to 3 (cores for CPU and GB for memory); therefore, there are a total of 81(3^4) resource configurations.

To have a direct feeling for the relationship between application performance and cost, we conduct experiments with all 81 application samples for each benchmark, shown in Figure 2. The cost adopted here is calculated based on a pricing rule of AWS, i.e., 0.33 unit/h for CPU and 0.20 unit/h for memory, respectively. It is obvious that the benchmark performance and cost vary greatly, and scatters are basically distributed in three regions (named as low, middle, and high, respectively). Taking the TeraSort as an example, the resource configuration with the minimal cost in high region is 0.0619 units with (1 $\text{memory}_{\text{map}}$, 1 CPU_{map}, 1 $\text{memory}_{\text{reduce}}$, 3 $\text{CPU}_{\text{reduce}}$), which is denoted by (1, 1, 1, 3).[15] Although the configuration with the maximal cost in low region is 0.1710 units with (3, 3, 3, 2), which means under an inappropriate allocation, applications will reach lower performance even with higher costs. Therefore, for a given performance, there are several resource configurations that can reach the target with different costs.

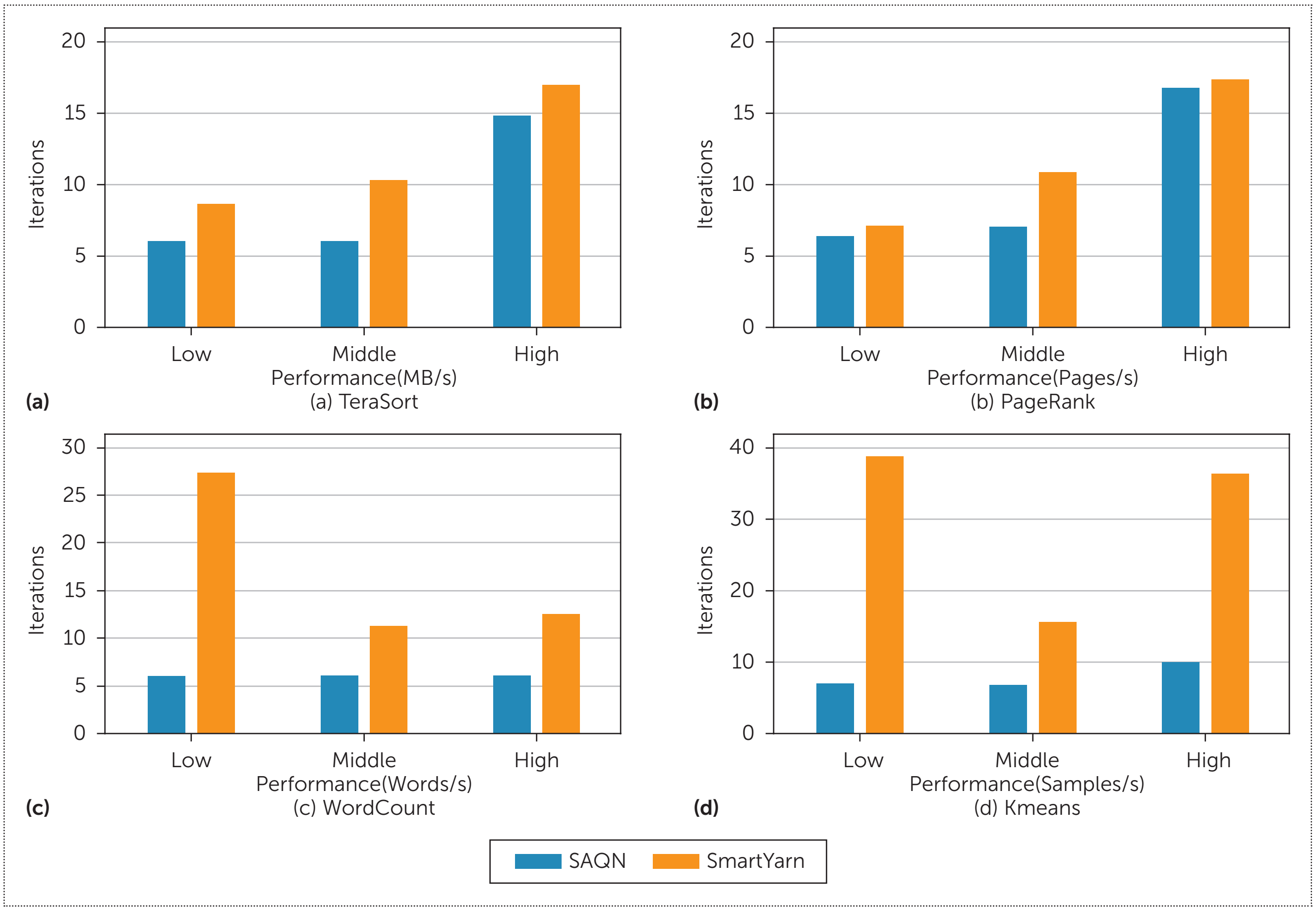

FIGURE 4. Efficiency comparisons between SAQN and SmartYARN.

To evaluate the accuracy of the proposed SAQN algorithm, we analyze the average cost reduction of the optimal configuration policy, SAQN and SmartYARN,[15] which is a recent intelligent resource management approach using fundamental reinforcement learning to negotiate the (near-) optimal policy, mainly for low-dimensional data, as follows:

$$\text{cost reduction ratio}_{\text{optimal}} \leftarrow \frac{\text{cost}_{\text{real}}^{\max} - \text{cost}_{\text{real}}^{\min}}{\text{cost}_{\text{real}}^{\max}}, \tag{3}$$

$$\text{cost reduction ratio}_{\text{SAQN}} \leftarrow \frac{\text{cost}_{\text{real}}^{\max} - \text{cost}_{\text{SAQN}}^{\min}}{\text{cost}_{\text{real}}^{\max}}, \tag{4}$$

$$\text{cost reduction ratio}_{\text{SmartYARN}} \leftarrow \frac{\text{cost}_{\text{real}}^{\max} - \text{cost}_{\text{SmartYARN}}^{\min}}{\text{cost}_{\text{real}}^{\max}}. \tag{5}$$

We can observe the results of the optimal policy and SAQN are the same in most situations and very close in others and shows the remarkable ability of SAQN in learning the (near-) optimal resource configuration. Besides, by taking advantage of all the reduction results in Figure 3, SAQN can reach 99.21 percent cost reduction of the optimal configuration policy in average, while SmartYARN can reach 98.13 percent, which shows the excellent ability of SAQN in processing low-dimensional data.

To verify the efficiency of SAQN, we compare the iterations among SAQN, SmartYARN, and exhaust algorithm, which can find the optimal resource configuration by exhausting all possible situations. Obviously, the exhaust algorithm needs 81 iterations to reach the optimal result. As shown in Figure 4, SmartYARN needs 15.89 iterations on average, while SAQN is able to learn the (near-) optimal configuration policy in only 8.07 iterations on average, showing its superior ability in online deployment.

In conclusion, SAQN can always provide the (near-) optimal configuration policy meeting the given demands through a limited number of learning processes, which is 8.07 iterations on average. As the main overhead of DRL is the cost spent on searching for the optimal policy, it is clear DRL is a practical approach in resource management. Compared with offline algorithms, SAQN shows its extraordinary ability of negotiating a more superior configuration policy, as it uses real-time measurements to optimize the action-value function, and compared with other online algorithms, SAQN exhibits its advantage in the reduction of iterations, especially for applications that need to run about several hours or even several days. Generally, SAQN is a better choice in resource management with multiple optimization goals, due to its combination of reinforcement learning to obtain the (near-) optimal policy directly from raw application inputs and deep learning to accelerate the learning process, especially for complex and high-dimensional data.

Future and Challenge

As a concluding remark, DRL is a new technology, but of great potential in resource management, due to its excellent ability of autonomic and efficient learning. Moreover, DRL shows its extensive applicability with the intelligent cloud, such as recommendations of transaction action for software as a service solutions, accurate discovery, and efficient organization of files for cloud storage service.

However, there are several challenging problems during the system deployment. First, parameters need to be carefully set to let DRL be most effective. Although not exhibited as difficult in this paper, as our experiments used low-dimensional data, generally, it is a very complicated and time-consuming process, especially for high-dimensional data. Besides, the environment assumption of DRL is based on MDP; however, there exist a number of situations that violate the assumption, and further research is needed to modify the DRL.

Given the extraordinary ability of DRL and the trend of the intelligent cloud, an automatic parameter tuner needs to be designed to facilitate the actual use. Moreover, the fields, such as hierarchical reinforcement learning aimed at solving semi-MDP, demand further research to flourish.

Acknowledgments

This work was supported, in part, by the Program for NSFC (no. 61572322 and no. 61525204), and the STCSM project (no. 16QA1402200).

References

1. Q. Zhang et al., "Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," *IEEE Trans. Cloud Computing,* vol. 2, no. 1, 2014, pp. 14–28.
2. X. Zhen, W. Song, and Q. Chen, "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment," *IEEE Trans. Parallel Distribution Systems,* vol. 24, no. 6, 2013, pp. 1107–1117.
3. W. Wang, B. Li, and B. Liang, "Dominant Resource Fairness in Cloud Computing Systems with Heterogeneous Servers," *Proc. 33rd Int'l Conf. on Computer Communications*, 2014.
4. J. Yao et al., "Control of Large-Scale Systems Through Dimension Reduction," *IEEE Trans. Services Computing,* vol. 8, no. 4, 2015, pp. 563–575.
5. A.S. Prasad and S. Rao, "A Mechanism Design Approach to Resource Procurement in Cloud Computing," *IEEE Trans. Computers,* vol. 63, no. 1, 2014, pp. 17–30.
6. L. Zhang, Z. Li, and C. Wu, "Dynamic Resource Provisioning in Cloud Computing: A Randomized Auction Approach," *Proc. 33rd Int'l Conf. on Computer Communications*, 2014.
7. R.H. Hwang et al., "Cost Optimization of Elasticity Cloud Resource Subscription Policy," *IEEE Trans. Services Computing,* vol. 7, no. 4, 2014, pp. 561–574.
8. Y. Duan et al., "Benchmarking Deep Reinforcement Learning for Continuous Control," *Proc. 33rd Int'l Conf. on Machine Learning*, 2016.
9. L. Deng, G. Hinton, and B. Kingsbury, "New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview," *Proc. 38th Int'l Conf. on Acoustics, Speech and Signal Processing*, 2013.
10. V. Mnih et al., "Human-Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, 2015, pp. 529–533.
11. Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, 2015, pp. 436–444.
12. S. Caton et al., "A Social Compute Cloud: Allocating and Sharing Infrastructure Resources via Social Networks," *IEEE Trans. Services Computing,* vol. 7, no. 3, 2014, pp. 359–372.
13. P. Vincent et al., "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *ACM J. Machine Learning Research,* vol. 11, 2010, pp. 3371–3408.
14. V. Mnih et al., "Asynchronous Methods for Deep Reinforcement Learning," *Proc. 33rd Int'l Conf. on Machine Learning*, 2016.
15. Y. Xu et al., "Cost-Efficient Negotiation over Multiple Resources with Reinforcement Learning," *Proc. 25th Int'l Symp. on Quality of Service*, 2017.

YU ZHANG *is working toward an MS degree, under the supervision of J. Yao, with the School of Electronic, Information and Electronic Engineering at the Shanghai Jiao Tong University, China. Her research interests are cloud computing, auto-configuration of virtual resources, and machine learning. Contact her at zhangy_se@sjtu.edu.cn.*

JIANGUO YAO *is an associate professor with the School of Electronic, Information and Electronic Engineering at the Shanghai Jiao Tong University, China. His research interests are cloud computing, auto-configuration of virtual resources, and virtualization technology. Contact him at jianguo.yao@sjtu.edu.cn.*

HAIBING GUAN *is a professor with the School of Electronic, Information and Electronic Engineering at Shanghai Jiao Tong University, China, and the director of the Shanghai Key Laboratory of Scalable Computing and Systems, China. His research interests are cloud computing, distributed computing, and virtualization technology. Contact him at hbguan@sjtu.edu.cn.*