A Workflow-Aided Internet of Things Paradigm with Intelligent Edge Computing

Yuwen Qian, Long Shi, Jun Li, Zhe Wang, Haibing Guan, Feng Shu, and H. Vincent Poor

Abstract

In this article, we propose a workflow-aided Internet of things (WIoT) paradigm with intelligent edge computing (IEC) to automate the execution of IoT applications with dependencies. Our design primarily targets at reducing the latency of the IoT systems from two perspectives. To reduce the latency from an application perspective, we develop a WIoT paradigm to orchestrate various IoT applications in a programming way. To reduce the latency from a computation perspective, we propose a novel IEC framework to execute latency-sensitive IoT tasks at the edge network. We put forth a deep reinforcement learning algorithm to adaptively allocate the edge resources to the dynamic requests, aiming to provide the best quality of service for terminal users in real-time. Furthermore, we design a software platform to implement the proposed WIoT with IEC. Experimental results demonstrate that WIoT with IEC can significantly reduce the service latency and improve the network throughput, compared with the traditional cloud-based IoT systems.

Introduction

Internet of Things (IoT) is a technical revolution that will bring us to the new era of many attractive applications, such as smart city, smart home, and smart grid, by interconnecting heterogeneous wireless sensors via the Internet [1]. Many IoT applications are computationally intensive, which are composed of hundreds and thousands of interrelated tasks. Workflow has recently been applied to the area of wireless IoT networks, named as workflow-aided IoT (WIoT). Different from the traditional approaches that deal with unrelated tasks, WIoT organizes IoT tasks with dependencies and enables automated cooperation among these tasks.

The cloud-based workflow [2], driven by centralized computational power of cloud computing, is one of the solutions to alleviate the computational cost of WIoT devices in an efficient way. In the cloud-based workflow paradigm, the cloud platform serves the requests combined by the workflow with a shared infrastructure. The resources in different geographical locations are first organized into a virtual resource pool. Then the virtual machine (VM) instances are allocated to the workflow tasks based on different service requirements, for example, minimizing the end-to-end delay [3], energy assumption [4], or total resource renting costs [5]. Like most cloud-based applications, the aforementioned literature mainly suffers from two limitations. First, some private and context-aware information, for example, precise user location, network state, or driving behavior of users, may not be directly available at the cloud, which reduces the overall resource allocation efficiency. Second, the application of cloud computing is constrained by limited bandwidth and cannot support latency-sensitive data streaming services.

An effective solution to reduce the latency of the cloud-based IoT applications is edge computing (EC) which takes advantage of ever-increasing computational and storage capabilities of the sensing and networking devices in the edge network [6, 7]. For example, the EC paradigm with a task offloading scheme in [8] not only obtains good adaptability and security, but also achieves high prediction accuracy and low processing delay. Furthermore, the energy consumption issue for the EC paradigm is examined in [9] where the task accomplishing energy can be effectively reduced by adopting the joint optimization scheme. By storing and executing applications locally rather than in the cloud, the EC-based IoT systems can reduce the data transmission delays and improve the private dataset protection. Some recent works in [10–12] combine the advantages of both cloud and EC for scientific workflow applications. A preprocessing method based on a genetic algorithm and particle swarm optimization is proposed in [10] to optimize the data placement strategy that minimizes the data transmission time across different cloud and edge data-centers during workflow execution. A cost-effective data replica placement strategy based on a variant of intelligent swarm optimization is proposed in [11] for reducing access costs under the IoT workflow deadline constraint in a collaborative edge and cloud computing environment. In [12], the authors put forth an energy-efficient workflow task scheduling algorithm to find the optimal communication start time among the cloud/edge computing nodes, aiming at minimizing the network energy consumption while satisfying user deadline constraints. However, the existing EC-based WIoT approaches in [10, 11, 12] mainly focus on optimizing the workflow structure for a specific type of IoT task (e.g., scientific computing) with the static user demands, where the system inputs remain unchanged during the whole

Digital Object Identifier:
10.1109/MNET.001.1900665

Yuwen Qian, Long Shi, Jun Li, and Zhe Wang are with Nanjing University of Science and Technology; Feng Shu is with Hainan University; Haibing Guan is with Shanghai Jiao Tong University; and H. Vincent Poor is with Princeton University. The corresponding authors are: Zhe Wang and Jun Li.

0890-8044/20/$25.00 © 2020 IEEE

network realization. In practice, since the user requests usually arrive dynamically, the system states (e.g., workflow queue lengths) are varied and correlated across time slots. In this case, the resource allocation cannot be simply solved by the traditional optimization-based approaches (e.g., like particle swarm or generic algorithms in [10, 11, 12]). To our best knowledge, our article is the first attempt to deal with dynamic resource allocation for EC-based WIoT by utilizing deep reinforcement learning (DRL). To cope with the temporal correlated system states, we formulate the problem as a Markov decision process (MDP) and adopt DRL to find the optimal resource allocation that minimizes the average latency in a complex network.

In this article, we propose a novel WIoT-aided intelligent EC (IEC) paradigm to automate the dynamic IoT task execution in real-time. We propose a three-layer framework, where the workflow layer bridges between dynamically arrived IoT application requests in the IoT application layer and dynamic availability of edge resources in the computing layer. Based on the proposed framework, we then illustrate how the proposed scheme deals with the latency issue from two perspectives. First, to reduce the latency from an application perspective, we adopt workflow to orchestrate different IoT applications in a programming way. In the proposed scheme, the workflow client devises the logics that describe the internal dependency of the IoT requests from the terminal users (TUs). Moreover, the workflow engine executes these requests automatically according to the TU-devised logic with a queueing system. Second, to reduce the latency from a computation perspective, the cooperative edge servers adaptively allocate the edge computing resources to the workflow request queues. We formulate this problem as an MDP and propose a DRL algorithm to find the optimal VM allocation policy that minimizes the expected network delay in terms of average queue length. Finally, the experimental results show that the proposed scheme can greatly reduce the network latency and increase the throughput, compared with other benchmark schemes.

Architecture of WIoT with IEC

To facilitate the automated execution of the IoT applications with dependencies, we propose a three-layer WIoT with IEC architecture including IoT application layer, workflow layer, and computing layer, as shown in Fig. 1. The workflow layer bridges between the dynamically arrived IoT application requests in the IoT application layer and the dynamic availability of the edge resources in the computing layer.

IoT Application Layer

The IoT application is designed to orchestrate the IoT applications involved in different requests of TUs. First, each TU establishes a workflow instance by arranging a set of IoT requests (e.g., J_1, J_2, ··· in Fig. 1) in the customized order on the WIoT application platform installed on the terminal devices (e.g., mobile phones and personal digital assistants), where these requests will be triggered automatically later according to this predetermined order. Second, the TUs submit their workflow instances to edge servers deployed in edge networks for the corresponding IoT services.

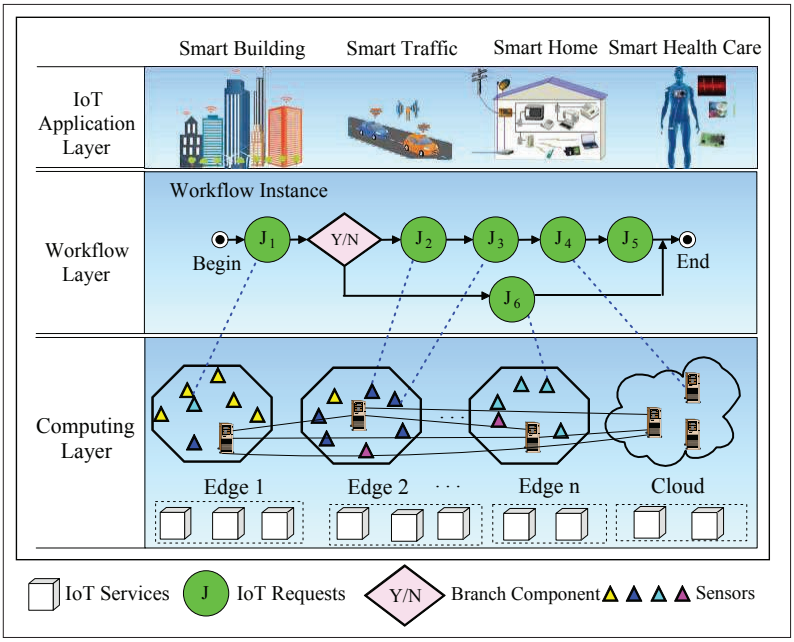

FIGURE 1. The architecture of WIoT with IEC, where J_1, ···, J_6 in the workflow instance are the IoT requests.

Workflow Layer

The workflow layer, located at the edge server, can parse the workflow instances and divide them into different requests, where each corresponds to a specific IoT service executed by the EC or cloud. Upon receiving the workflow instances, the edge servers parse the instances to extract useful information, and then negotiate with each TU to decide whether the workflow instance can be executed according to the budgets on the edge resources. Finally, the edge servers cooperatively schedule resources in different edge networks and the cloud to execute these requests.

Note that WIoT with IEC executes a workflow instance in two stages, that is, the building-time and running-time stages. In the building-time stage, the design of workflow instances adopts visual user-friendly programming tools, such as button, list view, and tree view. In particular, we devise the interface of workflow instance as a web client that can be edited with the browsers. In the running-time stage, the edge servers first check whether to accept the workflow instance according to edge resources and computational power. If the workflow instance is accepted, the edge servers execute the workflow instance.

Computing Layer

In this layer, computations are performed by the cooperation of the distributed edge nodes and the centralized cloud. According to the emergency level claimed by the user ID, the delay-sensitive computing and sensing requests are dispatched to the corresponding edge servers and sensor clusters. The routing delay is greatly reduced by processing the intensive computational tasks at the edge nodes rather than the cloud. In addition, EC can also analyze the raw data generated by

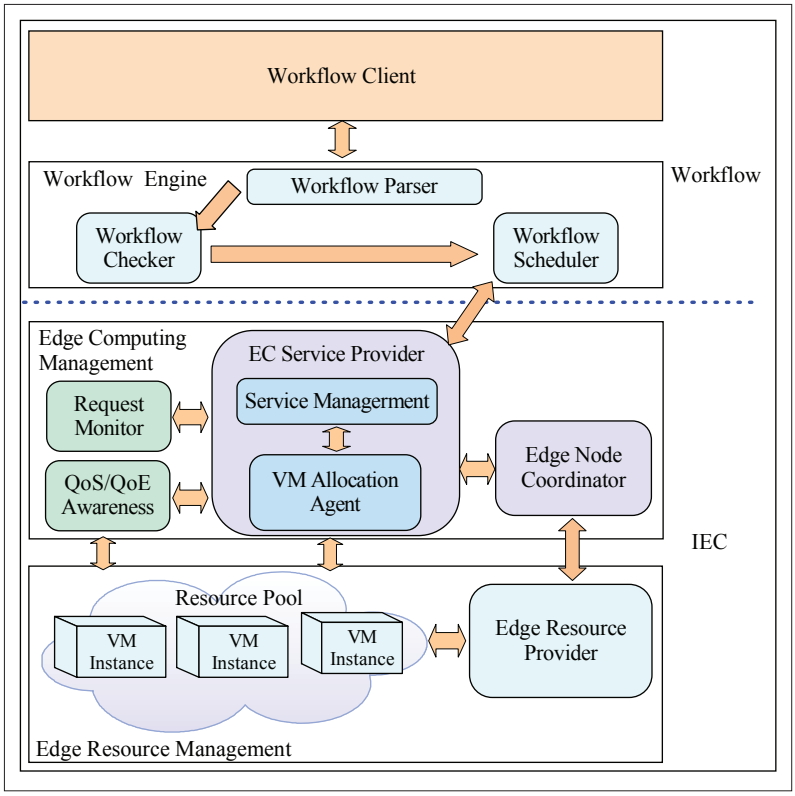

FIGURE 2. The software platform of WIoT with IEC used to implement the architecture in Fig. 2, where the workflow client used by the TUs organizes the IoT applications in the IoT application layer, the workflow engine automates the IoT applications in the workflow layer, and the IEC is the combination of EC management and edge resource management in the computing layer.

various sensors and provide timely response to the IoT applications. Moreover, the delay-tolerant input requests are executed by the cloud to save the computational costs.

Workflow Platform for IoT Systems

In this section, we develop a software platform to implement the proposed WIoT with IEC by using a client-server mode according to the architecture described previously. As shown in Fig. 2, the software platform consists of four core components: workflow client, workflow engine, EC management, and edge resource management.

Workflow Client

The workflow client provides the interface of workflow instance through a set of visual graphical user interface (GUI) tools, by which the TUs can interact with the edge networks. In particular, different clients are customized for specified terminal devices, such as web client with XML for personal computers and android client for mobile phones. Toward this end, the client employs basic programming components, that is, sequence, branch, and cycle, along with additional logic components such as time control, calendar, and parallel control components. Furthermore, network communication protocols such as TCP, HTTP, or SSH are inherently implemented in the workflow client. Add to that, there is a set of complementary tools including the workflow service image, alarm, auto-scaling, and billing components. Leveraging these tools, the TUs can run, halt, and monitor a workflow instance.

Workflow Engine

The workflow engine is used to schedule the requests from TUs to be executed. Figure 3 shows the structure of the workflow engine consisting of three primary modules: workflow parser, workflow checker and workflow scheduler. In addition, the requests are stored in different queues according to their states such as running, error, waiting, ready, and archived. To be specific, we design five queues for the requests according to their states, listed as follows.

- Running State: A request is in the running state if the request obtains VM instances and other required resources to run the corresponding IoT services.
- Ready State: A request is in the ready state if the request obtains all required resources except the VM instances.
- Waiting State: A request is in the waiting state if it has not obtained either the VM instances or some required resources.
- Error State: A request is in the error state if errors occur when any service associated with the request is running.
- Archived State: A request is in the archived state if all services associated with the request are accomplished.

Upon receiving the workflow instances from the clients, the request scheduling is elaborated in the following steps. First, the workflow parser analyzes the instances to obtain the corresponding requests and their descriptions. Second, the workflow checker checks authority and resource requirement of the requests. Third, the workflow scheduler schedules the request queues according to the executed results from the IEC system.

Workflow Parser: The workflow client adopts the serializing approach to convert workflow components, parameters, and data in each instance into character strings and then transmits them to the workflow engine.

After the un-serializing operation, the workflow engine can reconstruct the same requests as that in the workflow clients. By analyzing the descriptions of the requests, the engine obtains the specified requirements such as total completion time, budget for VM instances, and other required resources. Furthermore, the workflow engine calls out the corresponding IoT services based on the request descriptions.

Workflow Checker: After parsing, the workflow engine evaluates the instances by the following steps. The first step is to justify the authority of requesting TUs. The second step is to check the availability of IoT services and identify whether the service is available. The third step is to check the availability of computing resources that can be used to meet the requirements.

Workflow Scheduler: The workflow engine schedules the requests in workflow instances by operating the request queues. In WIoT with IEC, we adopt the network queue system (NQS) to manage the request queues [13]. The NQS can provide a complete queueing system to support remote queueing, routing, and access controls of both batch and device requests for a collection of machines in the network.

The workflow engine is also equipped with some functional components. First, the failover component can recover the requests in error state. After the recovery, the workflow engine resumes executing the corresponding service. Second, the load balancing component is to balance the number of requests over different edge servers. Third, the alarm component can generate alert information during the service processing.

Distributed IEC Computing

As shown in Fig. 2, we implement the workflow engine, IEC, and DRL-based VM instance allocation algorithm at the edge servers. The EC management collaborated with the edge resource management enables the IoT services corresponding to the requests in the workflow instances.

IEC System

The IEC system is composed of a group of scalable virtualized service components that collaborate to implement the VM instances. These VM instances are allocated to different requests in each workflow instance. Assigned with a sufficient number of VM instances, each IoT service involved in each request can be executed and the processing results are returned to both TUs and the workflow engine. In the following, we put forth an EC framework with a group of modules.

Request Monitor Module: The request monitor module (RMM) is used to offload the computation from the cloud to the edge. First, the RMM determines whether the submitted requests in the ready state from the workflow engine need to be executed to the cloud. If the request is delay-tolerant, the RMM migrates the request to the cloud for better computational resources. Otherwise, the RMM hosts the computation locally and creates the context for the request mainly including request ID, user ID, and request type. Second, the requests are inserted into the request queues, such that the edge resource provider can determine the number of VM instances in light of the context of requests. Notably, the RMM queues the ready-state requests according to their requested physical resources Third, the historical processing results and VM utilization results are memorized in the database.

Quality of Service (QoS) and Quality of Experience (QoE) Awareness Module: The QoS/QoE awareness module (QAM) is applied to monitor the QoS of IoT services. For simplicity, we use latency as a performance metric for both the QoS and QoE of the IoT requests, where response time is regarded as the QoE metric of each request and average response time is used as the QoS metric of the network. In particular, we handle the QoE fairness via a queueing method as follows. The user requests are classified into queues with different priorities, where the delay-sensitive requests are inserted into higher-priority queues and the delay-tolerant requests are plugged into low-priority queues. For the requests with the same priority, the first-in-first-out strategy is adopted to ensure the fairness among the requests. For the requests with different priorities, the system allocates more computing resources to the higher priority requests in order to meet more stringent delay requirements. In addition, the QAM can communicate with the VM allocation agent and service management module to monitor the VM instance queue, which is created by the edge resource provider module (Fig. 2). As a result, the QAM can estimate the response time and the accomplished service numbers per second for the TUs.

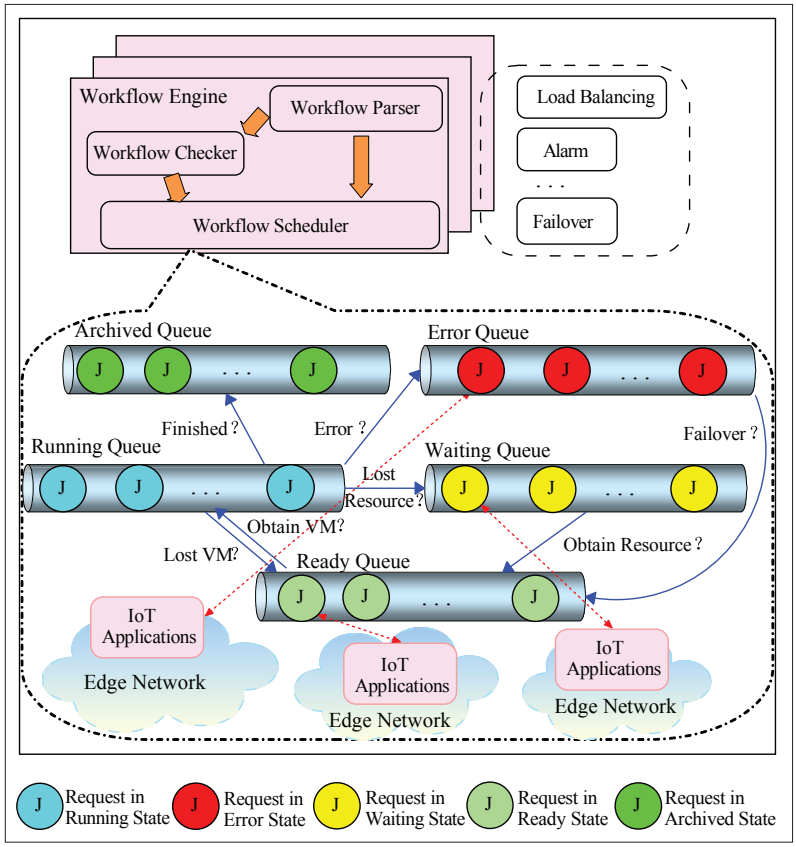

FIGURE 3. The structure of the workflow engine and the request scheduling in the workflow scheduler.

Service Management Module: The service management module (SMM) is developed to efficiently manage the IoT services at the edge. Initially, the SMM registers new services for IoT applications by the service registration function. Then, the SMM identifies the proper services for the requests queued by the RMM according to the registered service ID and the requests' description. Furthermore, the SMM can estimate the number of VM instances used to execute the request.

VM Allocation Agent: According to the VM estimation in the SMM, the VM allocation agent (VAA) assigns different amounts of VM instances to the corresponding services of the requests in the ready queues. In this article, we propose a DRL algorithm to address the VM allocation problem, aiming to reduce the response time and increase the number of accomplished services. As a result, the resource allocation decisions are forwarded from the VAA to the RMM and stored in the database.

Edge Node Coordinator: The edge node coordinator (ENC) collects the creation/termination information of VM instances from different edge servers. Then, the ENC sends the numbers of VM instances at different edge servers to the VAA for the purpose of load balancing. Here, we apply the load balancing approach to balance the VM instance numbers at different edge servers,

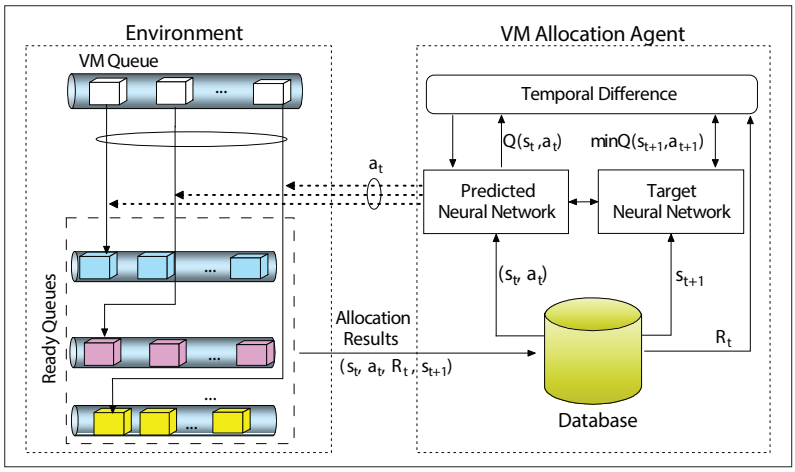

FIGURE 4. The structure of DRL-based VM allocation algorithm. The system is modeled as an MDP with parameters s_t, a_t, R_t, s_{t+1}, a_{t+1}, where st and at are the state and action in the current time slot, R_t is the rewarding function, and s_{t+1} and a_{t+1} are the state and action in the next time slot.

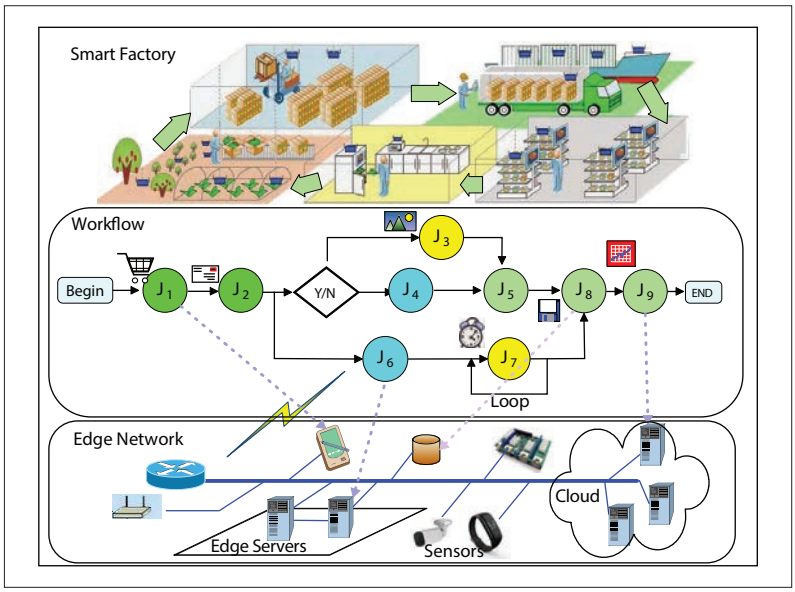

FIGURE 5. The implementation of WIoT with IEC, where the IoT requests in the workflow instance include contract creation J_1, sale approval J_2, urgent delivery J_3, normal delivery J_4, review collection J_5, package tracking J_6, payment reminder J_7, transaction information storage J_8, and analysis of transaction results J_9.

aiming to improve the computing efficiency. Furthermore, the ENC chooses proper edge servers to process the requests by cooperating with the nearby sensors.

In addition, the edge resource provider (ERP) supports the provision and deprovision of physical resources including edge servers, sensor clusters, and smart devices. First, the edge servers create the access points for the TUs and the smart devices. Second, according to the number of VM instances estimated in the SMM, the ERP creates the VM instances with the physical EC resources. Then, each VM instance is inserted into a VM instance queue created by the NQS.

Proposed DRL-Based VM Allocation

As shown in Fig. 4, the environment is comprised of the ready-state requests and VM queues. The VAA can observe the environment and adapt its decisions to the dynamic environment. Since the lengths of the queues are correlated across time, the resource allocation cannot be efficiently solved via traditional optimization approaches in each time realization. To address this issue, we first formulate the problem as an MDP to address the temporal correlation, and then develop a DRL-based allocation scheme in the VAA to dynamically assign a proper number of VM instances to each request.

MDP Formulation: In this section, we formulate the VM resource allocation as an MDP by defining the system state, action, and reward function, respectively. In addition, the running period is divided into T time slots, denoted by $t = \{1, 2, \ldots, T\}$. The duration of each time slot is D.

System State Space: We refer to the request in ready state as the ready-state request for brevity. Let k_t denote the number of ready queues at time slot t, m_t^k ($k \in \{1, \cdots, k_t\}$) denote the number of requests in the k-th ready queue at time slot t, and n_t denote the number of VM instances available in the VM queue at time t. Finally, the system state at time t is defined as $s_t = [m_t^1, \ldots, m_t^{k_t}, n_t]$.

Action Space: At the beginning of each slot t, we take the action a_t to allocate v_t out of n_t available VM instances to the k_t ready queues. Though some simple requests can be accomplished within one time slot, it may take multiple time slots to execute some complex requests. For normalization purposes, we divide each complex request into multiple request slices, where each request slice is assumed to be processed within one time slot. Thus, each queue k can be allocated with at most one VM instance in each time slot. In this context, $a_t^k \in \{0, 1\}$ and $v_t \le k_t$. If $a_t^k = 1$, the first request of the k-th queue will be processed and then inserted into the running queue in the workflow engine at the end of the time slot t. The action is finally defined as $a_t = [a_t^1, a_t^2, \ldots, a_t^{k_t}]$. Given that v_t VM instances are allocated, there are v_t processed requests/request slices in this time slot and the remaining requests are delayed. In the next slot $t + 1$, each state can be transferred to the other states according to certain transition probability.

Reward: The immediate reward R_t is defined as the total amount of delay (i.e., response time) induced by all unprocessed requests/request slices in the ready queues at time slot t, that is, $R_t = D[(\sum_{k=1}^{k_t} m_t^k) - v_t]$. For each time slot t, we aim at designing the optimal policy π^* (rule of selecting the actions) to minimize the expectation of the future delay, that is,

$$\min_{\pi} E\left[\sum_{i=t}^{T} \gamma^{i-t} R_i\right],$$

where $\gamma \in [0, 1]$ is the discount factor.

DRL-Based VM Allocation: Recently, a representative RL algorithm, for example, Q-learning, has been adopted to solve a variety of problems including path planning, routing design for mesh network, and marketing strategy selection. For the Q-learning algorithm, the key step is to design the state-action value function $Q(s_t, a_t)$, which can be consulted in a user-designed Q-table according to the current state and action. However, in WIoT with IEC, the Q-table design becomes prohibitively complicated as the states and actions proliferate. Driven by this issue, we employ a deep

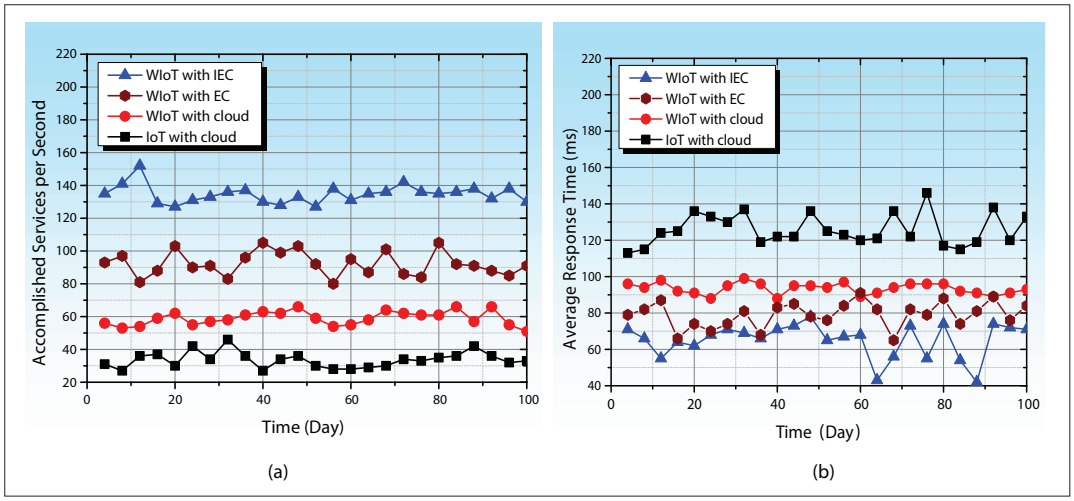

FIGURE 6. Experimental results comparison among the WIoT with IEC, WIoT with EC, WIoT with cloud, and IoT with cloud.

Q-network (DQN) algorithm to allocate the VM instances without the aid of Q-table, where useful information is trained by two neural networks to obtain the state-action value function $Q(s_t, a_t)$.

The proposed DRL algorithm is illustrated in Fig. 4. First, we employ the state-action value function $Q(s_t, a_t)$ to measure the profit induced by the state and action in the current time slot. Then, the VAA learns the rules from the environment, and updates $Q(s_t, a_t)$ by extracting mini-batch samples from the database and constructing the state-action pair (s_t, a_t). In particular, the DQN updates $Q(s_t, a_t)$ via the temporal difference between the outputs of the predicted neural network and the target neural network [14].

Finally, the VAA can identify the optimal policy to allocate the VM instances via DRL. The convergence proof for the proposed algorithm is similar to that of [14] and is omitted here.

Generally speaking, there exists a trade-off between AI/ML and lightweight edge computing. Due to limited computing resources, it is difficult for a lightweight server to execute complex computing services [8] and large-scale workflow. To solve this problem, we enable the cooperation among the edge servers in our IEC system. We first group the edge servers into clusters, and then select the edge server with the strongest computational power in each cluster as the edge cluster head to lead the local edge network consisting of nearby lightweight edge servers and embedded devices. By running the DRL algorithm at the edge cluster head, the computing resources of nearby servers (e.g., VM instances) are adaptively allocated to the workflow requests.

Implementation of WIoT with IEC

In this section, we implement the proposed WIoT with IEC in the system of an agricultural factory that produces, stores, sells, and delivers the products. Figure 5 illustrates the specific implementation of WIoT with IEC according to the three-layer framework in Fig. 1. First, automated logistics, smart agriculture, and product scheduling applications are involved in the IoT application layer.

In the second layer, we develop a typical workflow instance to automate the product transportation and selling services. First, a sale contract is created in J_1 upon receiving an order from the TUs, and the context of the contract is generated to record the order information. Second, J_2 is adopted to approve the contract. To be specific, the system first verifies the registered personal information, the price reasonability, and the availability of the products, and then submits them to the manager for sale approval. After the contract is approved, a parallel process with two branches is launched. In the first branch, the workflow engine decides whether the delivery is urgent. If yes, an express delivery in J_3 can be triggered. Otherwise, a normal delivery is activated in J_4. Once either J_3 or J_4 activates, the seller employs a courier delivery service company to deliver the products. When the customer receives the products, the customer's reviews are collected in J_5. In the second branch, J_6 tracks the product delivery. Subsequently, J_7 reminds customers to pay for the products. Upon receiving the payment and reviews from the customer, J_8 stores the transaction information. Finally, the transaction information is analyzed in J_9.

The third layer focuses on the implementation of the edge computing with the edge servers, sensors, and smart devices. Also, the EC management and edge resource management are implemented at the edge servers. Based on the degree of delay tolerance, each request queue is assigned with an initial priority value $n \in \{1, \cdots, N\}$, where 1 is the highest priority and N is the lowest. In each time realization, the available network resources (e.g., VM instances) are first allocated to the highest priority queue. In the cases of a tie (i.e., more than one queue has the highest priority), we randomly select one from these queues to process. Once a queue is being processed, its priority value will be reset to the initial priority value for the next time slot. Moreover, the priority value of all other unprocessed queues will be reduced by one. We repeat the above procedure throughout the time.

The experimental results are shown in Fig. 6. We deploy the proposed DRL algorithm in the edge server cluster head that has the highest computational power in each edge cluster. As benchmarks, we construct a typical IoT paradigm with cloud (IoT with cloud), a WIoT with cloud (WIoT with cloud) according to [15], and

a WIoT with edge computing (WIoT with EC). Note that different from the IEC scheme that minimizes the expected delay for all future time slots, the EC scheme only optimizes the VM allocation to minimize the delay for each time slot. In our experiments, we employ around 4,000 sensors to generate data over 100 days. The VM used in our experiment has one CPU unit, one CPU core, and 500M memory. All VM instances use an Ubuntu server as the operation system.

Figure 6a shows the numbers of average services per second accomplished by WIoT with IEC, WIoT with EC, WIoT with cloud, and IoT with cloud. We observe that the proposed WIoT with IEC accomplishes much more services than the other three schemes. For example, WIoT with IEC outperforms WIoT with EC and IoT with cloud by more than 30 percent and 70 percent, respectively.

Figure 6b demonstrates the average response time of the four schemes. First, we observe that IoT with cloud suffers from the largest response time among all schemes due to the lack of automated task execution aided by workflow. Second, the response time of WIoT with EC is lower than that with cloud, since a large number of the requests can be processed in the local edge networks. Third, WIoT with IEC consumes the lowest response time due to the adaptive resource allocation via DRL. Note that at several points, the performance gaps between WIoT with EC and IEC schemes are very small. This is due to the fact that the DRL targets at minimizing the expected delay in the long run (not limited to a day). Thus, it is possible that both EC and IEC schemes have similar performance in a single day. However, if we take the average over the whole time span, the IEC scheme significantly outperforms the EC scheme. Moreover, our experimental results show that it takes around an hour for the algorithm to converge on the first day, and the convergence speed is faster for the following days due to the well trained model obtained in the previous days. The details are omitted due to the limited space.

Conclusions

In this article, we first proposed a novel workflow-aided IoT paradigm with intelligence EC to provide IoT services by orchestrating different IoT applications. Benefiting from the EC, computing tasks can be immigrated from the cloud to the edge in close proximity to the sensors and TUs. Second, we designed a software platform of WIoT with IEC in a client-server mode, where workflow engine, EC management, and edge resource management are implemented in the distributed edge servers. Third, we developed a DQN-based algorithm to dynamically allocate the VM instances to the IoT services. Fourth, we implemented WIoT with IEC in an agriculture factory to produce, transport, and sell. Experimental results demonstrate that the proposed WIoT with IEC can substantially reduce the response time and accomplish much more services per second than some baseline schemes.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grants 61872184, 61727802, 61771244; in part by the National Key R&D Program under Grants 2018YFB1004800; and in part by the U.S. National Science Foundation under Grant CCF-1908308.

References

[1] E. Ahmed *et al.*, "Internet-of-Things-Based Smart Environments: State of the Art, Taxonomy, and Open Research Challenges," *IEEE Wireless Commun.*, vol. 23, no. 5, Oct. 2016, pp. 10–16.
[2] Y. Zhao *et al.*, "A Service Framework for Scientific Workflow Management in the Cloud," *IEEE Trans. Serv. Comput.*, vol. 8, no. 6, Dec. 2015, pp. 930–44.
[3] C. Wu *et al.*, "End-to-End Delay Minimization for Scientific Workflows in Clouds Under Budget Constraint," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, Apr. 2015, pp. 169–81.
[4] X. Xu *et al.*, "EnReal: An Energy-Aware Resource Allocation Method for Scientific Workflow Executions in Cloud Environment," *IEEE Trans. Cloud Comput.*, vol. 4, no. 2, June 2016, pp. 166–79.
[5] X. Li and Z. Cai, "Elastic Resource Provisioning for Cloud Workflow Applications," *IEEE Trans. Automat. Engin.*, vol. 14, no. 2, Apr. 2017, pp. 1195–1210.
[6] X. Sun and N. Ansari, "EdgeIoT: Mobile Edge Computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, Dec. 2016, pp. 22–29.
[7] S. Guo *et al.*, "Trusted Cloud-Edge Network Resource Management: DRL-Driven Service Function Chain Orchestration for IoT," *IEEE Int. Things J.*, vol. 7, no. 7, July 2019, pp. 6010–22.
[8] H. Guo, J. Liu, and J. Lv, "Toward Intelligent Task Offloading at the Edge," *IEEE Network*, vol. 34, no. 2, Apr. 2020, pp. 128–34.
[9] H. Guo and J. Liu, "UAV-Enhanced Intelligent Offloading for Internet of Things at the Edge," *IEEE Trans. Ind. Informat.*, vol. 16, no. 4, Apr. 2020, pp. 2737–46.
[10] B. Lin *et al.*, "A Time-Driven Data Placement Strategy for a Scientific Workflow Combining Edge Computing and Cloud Computing," *IEEE Trans. Ind. Infomat.*, vol. 15, no. 7, Jul. 2019, pp. 4254–65.
[11] Y. Shao, C. Li, and H. Tang, "A Data Replica Placement Strategy for IoT Workflows in Collaborative Edge and Cloud Environments," *Comput. Netw.*, vol. 107, no. 11, Nov. 2019, pp. 2204–39.
[12] X. Tang, W. Shi, and F. Wu, "Interconnection Network Energy-Aware Workflow Scheduling Algorithm on Heterogeneous Systems," *IEEE Trans. Ind. Informat.*, vol. Early Access, 2019.
[13] Kingsbury Sterling Software, "The Network Queueing System," http: //gnqs.sourceforge.net/downloads/index.html, accessed on July 2017.
[14] M. G. Azar *et al.*, "Minimax PAC Bounds on the Sample Complexity of Reinforcement Learning with a Generative Model," *Machine Learning*, vol. 91, no. 3, 2013, pp. 325–49.
[15] Ali Company, "Elastic compute service," https://www.alibabacloud.com/product/ecs?spm=5176.8789780.1280361.9.342157a8RC5FwW, accessed on Mar. 2018.

Biographies

Yuwen Qian received the Ph.D. degree in automatic engineering from Nanjing University of Science and Technology, Nanjing, China, in 2011. From July 2002 to June 2011, he was a lecturer in the Automation School of Nanjing University of Science and Technology. Since May 2019, he has been an associate professor in the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, China.

Long Shi [M'15] received the Ph.D. degree in electrical engineering from the University of New South Wales, Sydney, Australia, in 2012. From 2013 to 2016, he was a postdoctoral fellow at the Institute of Network Coding, Chinese University of Hong Kong, China. From 2014 to 2017, he was a lecturer at Nanjing University of Aeronautics and Astronautics, Nanjing, China. From 2017 to 2020, he was a research fellow at Singapore University of Technology and Design. He is now a professor at the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing, China.

Jun Li [M'09, SM'16] received the Ph.D. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, P. R. China in 2009. From January 2009 to June 2009, he worked in the Department of Research and Innovation, Alcatel Lucent Shanghai Bell as a research scientist. From June 2009 to April 2012, he was a postdoctoral fellow at the School of Electrical Engineering and Telecommunications, the University of New South Wales, Australia. From April 2012 to June 2015, he was a research fellow at the School of Electrical Engineering, the University of Sydney, Australia. Since June 2015 he has been a

professor at the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing, China. He was a visiting professor at Princeton University from 2018 to 2019. His research interests include network information theory, game theory, distributed intelligence, multiple agent reinforcement learning, and their applications in ultra-dense wireless networks, mobile edge computing, network privacy and security, and industrial Internet of things. He has co-authored more than 200 papers in IEEE journals and conferences, and holds one U.S. patent and more than 10 Chinese patents in these areas. He served as an editor for *IEEE Communication Letters* and he was a TPC member for several flagship IEEE conferences. He received the award as an Exemplary Reviewer from *IEEE Transactions on Communications* in 2018, and the best paper award from the IEEE International Conference on 5G for Future Wireless Networks in 2017.

ZHE WANG [M'14] received the Ph.D. degree in electrical engineering from The University of New South Wales in 2014. From 2014 to 2020, she was a research fellow with The University of Melbourne, and Singapore University of Technology and Design. She is currently a professor with the School of Computer Science and Engineering at Nanjing University of Science and Technology, Nanjing, China. Her research interests include applications of optimization, game theory, and machine learning to resource allocation in communications and networking.

HAIBING GUAN received the Ph.D. degree from Tongji University in 1999. He is currently a professor at the School of Electronic, Information and Electronic Engineering, Shanghai Jiao Tong University, and the director of the Shanghai Key Laboratory of Scalable Computing and Systems. His research interests include distributed computing, network security, network storage, green IT and cloud computing.

FENG SHU received the Ph.D., M.S., and B.S. degrees from the Southeast University, Nanjing, in 2002, XiDian University, Xi'an, China, in 1997, and Fuyang Normal University, Fuyang, China, in 1994, respectively. From October 2003 to October 2005, he was a post-doctor researcher with the National Key Mobile Communication Lab at the Southeast University. From September 2009 to September 2010, he was a visiting post-doctor at the University of Texas at Dallas. In October 2005, he joined the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing, China, where he is currently a professor and supervisor of Ph.D. and graduate students.

H. VINCENT POOR [S'72, M'77, SM'82, F'87] received the Ph.D. degree in EECS from Princeton University in 1977. From 1977 until 1990, he was on the faculty of the University of Illinois at Urbana-Champaign. Since 1990 he has been on the faculty at Princeton, where he is currently the Michael Henry Strater University Professor of Electrical Engineering. During 2006 to 2016, he served as Dean of Princeton's School of Engineering and Applied Science. He has also held visiting appointments at several other universities, including most recently at Berkeley and Cambridge. His research interests are in the areas of information theory, signal processing and machine learning, and their applications in wireless networks, energy systems and related fields. Among his publications in these areas is the recent book *Multiple Access Techniques for 5G Wireless Networks and Beyond* (Springer, 2019). He is a member of the National Academy of Engineering and the National Academy of Sciences, and is a foreign member of the Chinese Academy of Sciences, the Royal Society, and other national and international academies. Recent recognition of his work includes the 2017 IEEE Alexander Graham Bell Medal and a D.Eng. honoris causa from the University of Waterloo awarded in 2019.