ThriftyEdge: Resource-Efficient Edge Computing for Intelligent IoT Applications

Xu Chen, Qian Shi, Lei Yang, and Jie Xu

Abstract

In this article we propose a new paradigm of resource-efficient edge computing for the emerging intelligent IoT applications such as flying ad hoc networks for precision agriculture, e-health, and smart homes. We devise a resource-efficient edge computing scheme such that an intelligent IoT device user can well support its computationally intensive task by proper task offloading across the local device, nearby helper device, and the edge cloud in proximity. Different from existing studies for mobile computation offloading, we explore the novel perspective of resource efficiency and devise an efficient computation offloading mechanism consisting of a delay-aware task graph partition algorithm and an optimal virtual machine selection method in order to minimize an intelligent IoT device's edge resource occupancy and meanwhile satisfy its QoS requirement. Performance evaluation corroborates the effectiveness and superior performance of the proposed resource-efficient edge computing scheme.

Introduction

With the rapid development and widespread penetration of artificial intelligence technology, more and more novel intelligent Internet of Things (IoT) applications are emerging, such as deep-learning-driven smart video surveillance, flying ad hoc networks for precision agriculture, e-health, and smart homes [1]. These kinds of intelligent IoT applications are typically resource-hungry, run computationally intensive machine learning algorithms (e.g., deep learning [2]), and demand real-time processing [3]. Nevertheless, due to the physical size constraint, IoT devices are in general constrained by limited computation resources, which make difficult to fulfill the quality of service (QoS) requirements of intelligent IoT applications [4].

Task offloading, a promising technology to augment device resources, has been widely considered in both academia and industry. In recent years, many researchers have focused on the mobile cloud computing solution in which mobile devices can offload computation-intensive data processing tasks to the resource-rich clouds for remote execution. However, this paradigm cannot support latency-sensitive IoT applications, due to the long network distance between IoT devices and clouds [5]. As an alternative, mobile edge computing (more generally fog computing) is an emerging paradigm that leverages a multitude of collaborative end-user and/or near-user devices to carry out a substantial amount of computation tasks from which they can each benefit [5]. As mobile edge computing is implemented at the network edge, it promises low latency as well as agile computation augmenting services for IoT device users [5, 6]. As illustrated in Fig. 1, mobile edge computing can enable a hybrid task offloading paradigm for intelligent IoT applications, where an IoT device is enabled to utilize its own local resource, nearby devices' resources via device-to-device (D2D) communication technology and the edge clouds' resources in proximity to the connected cellular base station.

There have been many existing studies considering the computation offloading problems, in particular for mobile cloud computing (e.g., [7–11, references therein]). Nevertheless, most of these studies only focused on the computation offloading between the target device and the associated cloud server. How to efficiently offload computation-intensive and and latency-critical intelligent IoT application tasks across the target device, helper devices, and the edge cloud is less understood. Moreover, due to the massive nature of IoT devices and the limited capacity of the edge cloud, concurrent computation offloading across different devices is highly significant and necessary. Hence, computation offloading for each individual IoT device should be resource-efficient subject to the satisfied QoS requirement. More resource-efficient computation offloading also implies less resource usage and hence a lower edge cloud service payment for the IoT device user. However, the resource efficiency of computation offloading is rarely explored in the existing studies.

With this motivation, in this article we propose ThriftyEdge, a new paradigm of resource-efficient edge computing for intelligent IoT applications. In particular, we adopt a device-centric approach, and devise a resource-efficient computation offloading algorithm such that each individual intelligent IoT device user strives to minimize the cloud resource occupancy (usage) given that its QoS requirement is satisfied. The salient features of our resource-efficient computation offloading algorithm are as follows. On one hand, we adopt a hybrid approach to exploit the hierarchical resources across local devices, nearby helper devices, and the edge cloud in proximity. On the other hand, we propose an efficient topology-sorting-based task graph partition algorithm

Xu Chen is with Sun Yat-sen University and Guangdong Key Laboratory for Big Data Analysis and Simulation of Public Opinion; Qian Shi is with Sun Yat-sen University; Lei Yang is with South China University of Technology; Jie Xu is with Guangdong University of Technology. The corresponding author is Qian Shi.

Digital Object Identifier: 10.1109/MNET.2018.1700145

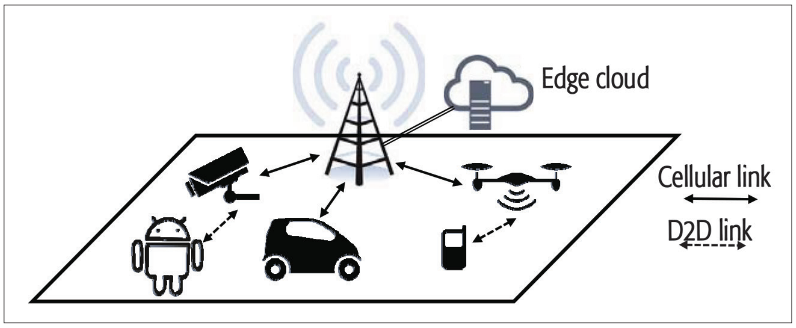

FIGURE 1. An illustration of mobile edge computing for intelligent IoT applications.

and the optimal virtual machine selection method in order to minimize an intelligent IoT device's edge resource occupancy and meanwhile satisfy its QoS requirement.

Related Work

Many previous works in mobile cloud computing have focused on the computation offloading problem (e.g., [7–11, references therein]). Wen *et al.* in [7] presented an efficient offloading policy by jointly configuring the clock frequency in the mobile device and scheduling the data transmission to minimize the energy consumption. Yang *et al.* in [8] studied the scenario in which multiple users share the wireless network bandwidth, and solved the problem of minimizing the delay performance by a centralized heuristic genetic algorithm. Chen *et al.* in [9] proposed a game theoretic approach for designing decentralized multi-user computation offloading mechanisms for reducing the total task execution overhead. Pu *et al.* in [10] devised a mobile edge computing framework based on D2D collaboration with the objective of optimizing the system-wide energy efficiency.

Note that the computation offloading scheme in our article is different from the studies above, since the existing works aim to optimize a device's energy efficiency or delay performance, while our objective is to optimize a device's offloading decision with respect to resource utilization efficiency (subject to completion time constraints). This is highly relevant to many IoT applications because the number of IoT devices is generally large, and it is desirable to support more concurrent IoT device users by minimizing each individual device's resource occupation.

Edge Computing Scenario

We consider a scenario in which an intelligent IoT device (e.g., a smart video camera or a robot) is connected to a wireless base station that possesses an edge cloud in proximity. There are multiple virtual machine types available at the edge cloud, each of which has heterogenous computation capability. Note that our model can also be extended to the case with the remote public cloud by just taking into account the wide area network (WAN) transmission latency.

The IoT device has a time-critical and computation-intensive task (e.g., a machine learning task for intelligent IoT applications) to complete. Subject to the QoS requirement of the application, the task has the maximum allowable completion time. When there is also a helper device (e.g., a device in an idle status) available in proximity, the task device can also choose to establish a D2D connection with it (e.g., using cellular D2D [12] or WiFi-Direct [13]) for D2D-enabled computation offloading.[1] This is motivated by the fact that the edge device types are in general heterogonous, and some devices (e.g., surveillance video camera and robots) have both sufficient battery power and computing capacity, and hence are capable of handling the offloaded tasks from a nearby device that is resource constrained. Note that when there is not such a helper device available, the task device only considers the choices of both local computing and offloaded edge cloud computing.

We next introduce the task graph model to describe the processing procedure of the computational task. Similar to most existing studies (e.g., [7, 8, 14]), we describe the data processing computation task of the device by a directed acyclic graph. Specifically, in the task graph, the nodes represent the data processing task components, and the directed edges stand for the data dependencies. For a node in the task graph, we can define the node weight as the required computational resources (i.e., CPU cycles) for the corresponding data processing task component. For a subsequent node that invokes the output data from another node, we can also define the edge weight between these two nodes as the amount of data transferred between these two corresponding task components.

For a given task component on the task graph, the task device can either execute it locally at its own device, or offload to the helper device in proximity or the edge cloud. Hence, we have the time for executing the given task component i at device m as T_i^m, which depends on the computation capability of the execution device. For the edge cloud computing approach, a virtual machine is associated with the task device user for its task execution in the edge cloud. Depending on the virtual machine type chosen by the device for executing its whole task, accordingly we can then compute the time by the cloud computing approach for executing the given task component i on the virtual machine as T_i^c.

Now suppose that two task components i, j having data dependency are executed at different locations (e.g., one at a user's device and the other at the edge cloud). Then a total amount of data needs to be transferred between these two components via wireless connections [12]. We denote the execution location of the task component i as y_i, and the total data transmission time for computation offloading between data processing task components i and j is represented as $T_{ij}^e(y_i, y_j)$.

Resource-Efficient Computation Offloading

In this section, we discuss the key idea of ThriftyEdge. We first consider the resource-efficient computation offloading problem to determine the optimal task component execution locations and the virtual machine selection in order to minimize the resource occupancy and meanwhile satisfy the QoS requirements.

[1] For ease of exposition, we only consider one helper device here. Nevertheless, our approach can easily be extended to the cases of multiple helper devices by just expanding the choice space for computation offloading.

Delay-Aware Task Graph Partition

We first consider the delay-aware task graph partition problem for a fixed edge cloud resource size. That is, given that the user's virtual machine type selection for cloud computation is fixed, we aim to determine the optimal execution location (i.e., target device n, helper device m, or edge cloud c) for each task component in its task graph in order to minimize the total delay of executing the whole task.

First of all, we leverage the structural property of the task graph. Since the graph is directed acyclic, it is known that we can carry out a topological sorting of the graph (Fig. 2 illustrates this), that is, ordering of the nodes such that for every directed edge from node i to node j, i comes before j in the ordering.

We introduce the classic Kahn Algorithm for the topological sorting of the task graph, which has a low polynomial computational complexity. After the topological sorting, we then obtain an ordering of the nodes. We also denote the set of nodes to which there is an edge directed from node i as $\Delta(i)$. According to the definition of topology sorting, we know that if $j \in \Delta(i)$, i comes before j in the node ordering. Note that since no edge is directed from the end node (i.e., the output component) and for any other node there always exists some edge directed from it (otherwise the corresponding task component is useless since it does not provide any input to either other intermediate components or the output component), the last node in the ordering after topology sorting must be the output component.

Based on the node ordering by topological sorting, we then solve the delay-aware task graph partition problem by the principle of backward induction. That is, we first determine the optimal execution location for the last node in the ordering after topology sorting, based on which we can determine the optimal location for the nodes by moving backward. Moreover, we denote the minimum delay starting from executing the task component i until the end of the whole task as $Z(i)$.

For the last node k (i.e., the output component) in the ordering, we have $Z(k)$ equals the execution time t_k^n at the local task device n, since in general the final result will be returned to the task device n. Then we move backward to consider the remaining nodes in the ordering in a recursive manner. Obviously, the delay starting from executing the task component i until the end of the whole task depends on that of the bottleneck node $j \in \Delta(i)$ with maximum delay that requires the input data from node i. Thus, we have $Z(i) = \min_{y \in \{n,m,c\}} \max_{j \in \Delta(i)} \{T_i^y + T_{ij}^e(y, y_j) + Z(j)\}$. Intuitively, this means that the minimum delay starting from executing the task component i until the end of the whole task equals minimizing the total delay of both executing the current task component (including the data transmission delay if any) and the minimum executing delay for the remaining task components afterward. Then the optimal execution location for a node is the best choice for achieving the minimum delay. By the back induction, for all the remaining nodes, we can recursively compute the minimum delays and the optimal execution locations.

The proposed algorithm consists of both a delay-aware task graph partition algorithm and the optimal virtual machine selection method, aiming at minimizing an intelligent IoT device's edge resource occupancy while satisfying the QoS requirement.

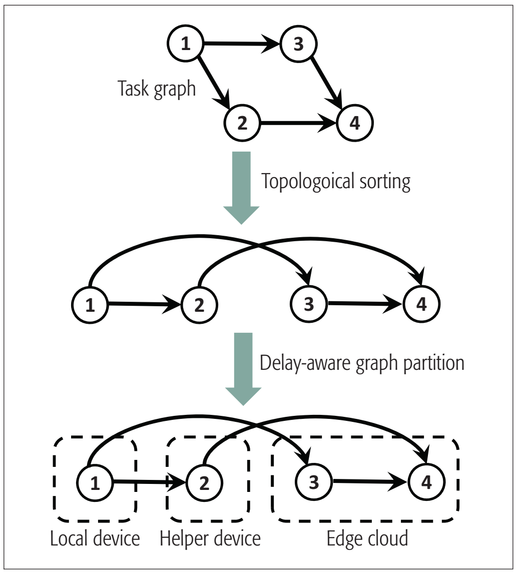

FIGURE 2. An illustration of task graph topological sorting and partitioning for mobile edge computation offloading.

Optimal Virtual Machine Selection

We aim to devise a resource-efficient computation offloading strategy in order to determine the optimal edge cloud resource profile for the device, with minimum edge cloud resource occupation while satisfying its QoS constraint. The key motivation is two-fold. On one hand, from the system point of view, minimizing cloud resource occupation helps to increase the resource utilization efficiency and can support more concurrent IoT devices in the mobile edge system. On the other hand, from the user perspective, in general the cloud resource occupancy serves as an important indicator for determining user payment for edge computing service. Thus, a smaller cloud resource occupancy implies a lower service payment for the user as well. Formally, we define the resource-efficient offloading problem (i.e., the core problem for ThriftyEdge) as selecting the proper virtual machine type that minimizes the edge cloud resource occupancy, subject to the constraint that the completion time of the task is within the deadline.

As illustrated in Fig. 3, building upon the delay-aware task graph partition algorithm above, this problem can easily be solved as follows. First, we rank all virtual machine types provisioned by the edge cloud according to the corresponding computation resource capacity. Then, according to the ranking, we sequentially select a virtual machine choice and utilize the delay-aware task graph partition algorithm to compute the minimum delay for the first node in the ordering after the topology sorting. Once the execution time is less than the deadline, we stop and find the optimal virtual machine choice having the minimum resource occupation. Note that for simplicity in this article

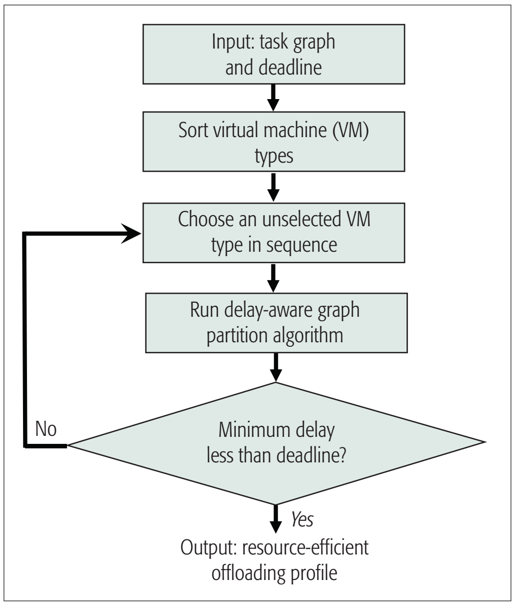
Input: task graph and deadline
Sort virtual machine (VM) types
Choose an unselected VM type in sequence
Run delay-aware graph partition algorithm
Minimum delay less than deadline?
No
Yes
Output: resource-efficient offloading profile

FIGURE 3. Flow chart of computing resource-efficient computation offloading profile

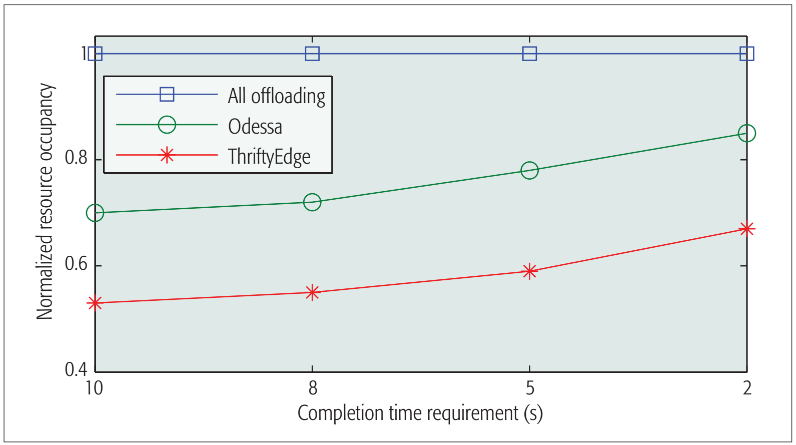

FIGURE 4. Resource occupancy by different computation offloading schemes.

we only consider the delay as the constraint. Our method can also apply if we consider the energy overhead. This can be done by accounting for the accumulated energy cost in the back induction procedure in the delay-aware task graph partition algorithm.

We further evaluate the performance of the proposed ThriftyEdge scheme (i.e., the resource-efficient offloading algorithm) in terms of the resource occupancy. Upon comparison, we also implement the Odessa computation offloading scheme [15] and the all offloading solution — all task components in the task graph will be offloaded. The Odessa computation offloading scheme adopts a greedy strategy to leverage the data and pipeline parallelism for computation offloading. The results are shown in Fig. 4 using object pose estimation application, wherein we use the all offloading solution as the baseline to normalize the resource occupancy performance by the other two schemes. We see that the proposed ThriftyEdge scheme can achieve superior performance of 19 percent resource occupancy reduction over the Odessa scheme.

Future Research Directions

In this section, we discuss several important directions for further exploring resource-efficient edge computing for intelligent IoT applications.

First of all, a very interesting and challenging direction for extending the current study is to consider multi-user resource-efficient edge computing. Different from the single-user case, for the multi-user case we need to carefully address the key contention issue in the computing resource sharing on both the same helper device and the edge cloud. Also, it is very useful to explore the reusability of the task components among multiple users that are running similar applications in order to further improve the resource efficiency.

Second, it is also very promising to design a fine-grained computation offloading mechanism, such that different IoT devices can fine tune their computation demands tailored to the physical environment as well as the QoS requirements. Achieving fine-grained computation offloading can definitely help to improve the resource efficiency.

Last but not least, in this article we consider a general computation model that is represented by a computational task graph. Since the IoT data analytics is of prime importance for many intelligent IoT applications, particular research attention can be paid to the area of edge data stream processing, which leverages both edge computing and machine learning techniques for achieving efficient real-time IoT data processing and analytics.

Conclusion

In this article we explore the resource-efficient edge computing issues for intelligent IoT applications. We devise a resource-efficient computation offloading algorithm to enable an IoT device to exploit resources across the local device, nearby helper devices, and the edge cloud in proximity. Specifically, the proposed algorithm consists of both a delay-aware task graph partition algorithm and the optimal virtual machine selection method, aiming to minimize an intelligent IoT device's edge resource occupancy while satisfying the QoS requirement. Performance evaluation corroborates the superior performance of the proposed computation offloading algorithm in terms of the resource efficiency.

Acknowledgment

This work was supported by the National Key Research and Development Program of China under grant No. 2017YFB1001703, the National Science Foundation of China under Grant Nos. U1711265, 61601522 and 61502312, the Fundamental Research Funds for the Central Universities under grant No.17lgjc40, and the China Postdoctoral Science Foundation under Grant No.2015M580753.

References

[1] A. Whitmore, A. Agarwal, and L. Da Xu, "The Internet of Things — A Survey of Topics and Trends," *Info. Systems Frontiers*, vol. 17, no. 2, 2015, p. 261.
[2] Z. Fadlullah *et al.*, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems," *IEEE Commun. Surveys & Tutorials*, 2017.

[3] N. Kato *et al.*, “The Deep Learning Vision for Heterogeneous Network Traffic Control: Proposal, Challenges, and Future Perspective,” *IEEE Wireless Commun.*, vol. 24, no. 3, June 2017, pp. 146–53.

[4] J. Ren *et al.*, “Serving at the Edge: A Scalable iot Architecture Based on Transparent Computing,” *IEEE Network*, 2017.

[5] A. Ahmed and E. Ahmed, “A Survey on Mobile Edge Computing,” *Int’l. Conf. Intelligent Systems and Control*, 2016, pp. 1–8.

[6] J. Ren *et al.*, “Exploiting Mobile Crowdsourcing for Pervasive Cloud Services: Challenges and Solutions,” *IEEE Commun. Mag.*, vol. 53, no. 3, Mar. 2015, pp. 98–105.

[7] Y. Wen, W. Zhang, and H. Luo, “Energy-Optimal Mobile Application Execution: Taming Resource-Poor Mobile Devices with Cloud Clones,” *IEEE INFOCOM*, 2012.

[8] L. Yang *et al.*, “A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, 2013, pp. 23–32.

[9] X. Chen *et al.*, “Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing,” *IEEE/ACM Trans. Net.*, vol. 24, no. 5, 2016, pp. 2795–808.

[10] L. Pu *et al.*, “D2D Fogging: An Energy-Efficient and Incentive-Aware Task Offloading Framework Via Network-Assisted D2D Collaboration,” *IEEE JSAC*, vol. 34, no. 12, 2016, pp. 3887–901.

[11] Y. Ai, M. Peng, and K. Zhang, “Edge Cloud Computing Technologies for Internet of Things: A Primer,” *Digital Commun. and Networks*, 2017.

[12] X. Chen *et al.*, “Exploiting Social Ties for Cooperative D2D Communications: A Mobile Social Networking Case,” *IEEE/ACM Trans. Net.*, vol. 23, no. 5, 2015, pp. 1471–84.

[13] M. Conti *et al.*, “Experimenting Opportunistic Networks with WiFi Direct,” *IFIP Wireless Days*, 2013.

[14] K. Kumar *et al.*, “A Survey of Computation Offloading for Mobile Systems,” *Mobile Networks and Applications*, 2013, pp. 1–12.

[15] M.-R. Ra *et al.*, “Odessa: Enabling Interactive Perception Applications on Mobile Devices,” *ACM Int’l. Conf. Mobile Systems, Applications, and Services*, 2011.

Biographies

Xu Chen received his Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012. He was a postdoctoral research associate with Arizona State University, Tempe, from 2012 to 2014, and a Humboldt Fellow with the University of Goettingen, Germany, from 2014 to 2016. He is currently a professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. He received the 2017 IEEE ICC Best Paper Award, the 2014 IEEE INFOCOM Best Paper Runner-Up Award, and the 2014 Hong Kong Young Scientist Runner-Up Award.

Qian Shi received her B.S. degree in sciences and techniques of remote sensing from Wuhan University, China, in 2010 and her Ph.D. degree in photogrammetry and remote sensing from the State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University in 2015. She is currently a postdoctoral fellow with the School of Geography and Planning, Sun Yat-sen University. Her research interests are in machine learning and image processing, including active learning, transfer learning, and dimension reduction for hyperspectral image.

Lei Yang obtained his PhD degree from the Department of Computing, Hong Kong Polytechnic University in 2014, his M.S. degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences in 2010, and his B.S. degree in electronic engineering from Wuhan University in 2007. He is currently an associate professor at the School of Software, South China University of Technology. Before that, he worked as a postdoctoral fellow at the Department of Computing, Hong Kong Polytechnic University. He was at Deeds Lab at Technische Universitat Darmstadt, Germany, as a visiting scholar from November 2012 to March 2013.

Jie Xu received his B.E. and Ph.D. degrees from the University of Science and Technology of China in 2007 and 2012, respectively. He is now a professor with the School of Information Engineering, Guangdong University of Technology, China. From 2012 to 2014, he was a research fellow with the Department of Electrical and Computer Engineering, National University of Singapore. From 2015 to 2016, he was a postdoctoral research fellow with the Engineering Systems and Design Pillar, Singapore University of Technology and Design. He is now an Editor for *IEEE Wireless Communications Letters* and an Associate Editor for *IEEE Access*.