DOI: 10.3969/j.issn.1673-5188.2017.04.005
http://kns.cnki.net/kcms/detail/34.1294.TN.20171023.1016.002.html, published online October 23, 2017

Online Shuffling with Task Duplication in Cloud

ZANG Qimeng[1] and GUO Song[2]
(1. School of Computer Science and Engineering, The University of Aizu, Aizu-Wakamatsu 965-0006, Japan;
2. Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR 852, China)

Abstract

Task duplication has been widely adopted to mitigate the impact of stragglers that run much longer than normal tasks. However, task duplication on data pipelining case would generate excessive traffic over the datacenter networks. In this paper, we study minimizing the traffic cost for data pipelining task replications and design a controller that chooses the data generated by the first finished task and discards data generated later by other replications belonging to the same task. Each task replication communicates with the controller when it finishes a data processing, which causes additional network overhead. Hence, we try to reduce the network overhead and make a trade-off between the delay of data block and the network overhead. Finally, extensive simulation results demonstrate that our proposal can minimize network traffic cost under data pipelining case.

Keywords

cloud computing; big data; shuffling; task duplication; traffic

1 Introduction

Driven by technology and economies of scale, cloud computing platforms are becoming the mainstream hosting platform for a variety of infrastructure services and data intensive applications [1]. With the development of cloud computing, large scale data computing has gained widespread attention due to its ability to automatically parallelize a job into multiple short tasks, and transparently deal with the challenge of executing these tasks in a distributed setting. The crux is that the execution time of a task is variable. The tasks on the slowest machines (stragglers) become the bottleneck in the completion of a job. Straggler mitigation has received considerable attention across prior studies [2], [3]. Launching copies for the slower tasks is the dominant technique to mitigate the stragglers. A fundamental limitation is that traditional techniques for straggler mitigation do not take into account of network traffic in the pipelining case. While this approach of replicating tasks decreases task completion time, it may increase network traffic. The wide area network (WAN) that connects geographically distributed machines is one of the most critical and expensive infrastructures that costs hundreds of millions of dollars annually [4].

Thus, providing traffic awareness for pipelining task duplications is essential for big data processing applications in cloud. To the best of our knowledge, however, no existing work is in place to respect the network traffic in the pipelining case. We seek for a mechanism to minimize the network traffic cost by coordinating task duplication and pipelined intermediate data.

In this paper, we introduce a controller to manage task duplications. Every task duplication has a communication link with the controller. To reduce network traffic, the controller picks the first finished task output from a duplication and discards the subsequent data generated later by other duplications. Each task duplication produces a large number of key-value pairs for pipelining data records. The communication between controller and each duplication will greatly increase the network overhead. To solve this problem, we propose an online distributed algorithm which is executed by each task duplication. The algorithm makes a rule to group data records and transmit the group instead of transmitting one data record at a time. This solution helps reduce the communication times with the controller, but may cause delay. We thus make a trade-off between the transmitting delay and communication overhead.

The remainder of the paper is structured as follows. Section 2 reviews the parallel computing and related work. Section 3 describes the motivation and challenges of the proposed work. Section 4 gives an overview of our system model and formulates the network traffic minimization problem. Section 5 introduces online distributed algorithms. Section 6 demonstrates the performance evaluation results. Finally, section 7 concludes this work.

2 Background and Related Work

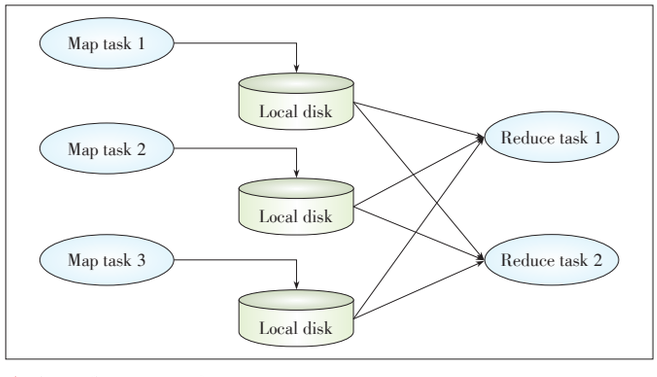
MapReduce [5] has emerged as the most popular program-

ming framework for big data processing in cloud. Ever since it was introduced by Google, the MapReduce programming model has gained in popularity, thanks to its simple, yet versatile interface. The paradigm has also been implemented by the open source community through the Hadoop project [6], maintained by the Apache Foundation. MapReduce divides a computing job into two main phases, namely map and reduce, which in turn are carried out by several map tasks and reduce tasks. The intermediate data transmission process from map task to reduce task is referred to as shuffling. We use a typical MapReduce job to show how the model works. As shown in **Fig. 1**, several map tasks are executed at the same time. The generated intermediate results in forms of key-value pairs are stored in local storage. To simplify fault tolerance, MapReduce materializes the output of each map before it can be consumed. A reduce task cannot fetch the output of a task until the map has finished executing and committed its final output to disk.

A fundamental limitation of MapReduce is that the intermediate data is materialized. Storing intermediate data on the local disk impacts MapReduce's performance. Thus, optimizing the intermediate data management is a popular topic in industry and academic research. In order to improve shuffling performance, a modified MapReduce architecture is proposed, MapReduce Online, in which intermediate data is pipelined between operators [7]. When a map task generates a key-value pair, it immediately sends this key-value pair to corresponding reduce task, avoiding intermediate data storage. More- over, after receiving key-value pairs, reduce tasks can start data processing earlier, without waiting map tasks to finish. Pipelining intermediate data improves system utilization.

The technique of replicating tasks has been widely applied in parallel computing by system designers [8].

By adopting task duplication, the distributed computing systems achieve predictable performance. Several large scale applications use the technique of task duplication to mitigate stragglers, such as Longest Approximate Time to End (LATE) [9], Dryad in Microsoft [10] and Mantri [11]. When the fastest task finishes, its output will be stored and then other running task copies will be killed. The original MapReduce paper handles the stragglers by adopting backup tasks.



▲**Figure 1. The MapReduce model.**

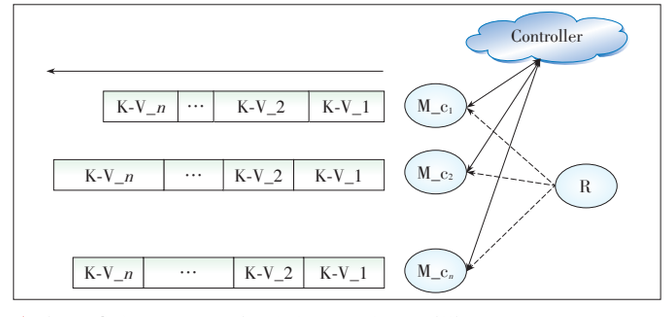
However, the traditional techniques of straggler mitigation causes excessive network traffic when the pipelining case is used for them. Replicating tasks in large scale applications has a long history [12], [13] with extensive studies in prior work. However, these studies are not applicable in the pipelining case. For example, we suppose that multiple task copies are launched for a map task with pipelined intermediate data. Whichever task copy produces intermediate data, the reduce task will receive the data, even though it has received the data from other task copy already, which generates huge data transmission in the shuffling process. The most important difference among our work and the related work is that our proposed method respects the network traffic. Since the limited bandwidth shared by multiple applications, the redundant data transmission would congest the network and bring negative performance for cloud computing applications. To fill in this gap, we take a stab at task duplication for pipelining data transfer.

3 Motivation

Here, we illustrate the value of task duplication in the pipelining case and give a reasonable solution.

When big data processing is executed on cloud, these jobs consist of many parallel tasks. Every task is executed in parallel on different machines. In our proposed work, there is a controller to manage the execution of each task replication. In the same way, every task replication runs in parallel on different machines. The controller will receive a notification when a machine finishes its assigned task, and then choose the first finished task to transmit the data. The data generated by other task replications will be discarded after the controller decides one of the task replications to transmit the data records. By adopting this method, the overlapping data will not be transmitted, which minimizes the network traffic cost generated by same duplications. As shown in **Fig. 2**, multiple task copies are launched for one of the map task, i.e. M_c1, M_c2 and M_c*n*. Each task copy produces the key-value pair in a different time. The controller will pick the fastest one for the input of the reduce task.

However, another crucial factor should be taken into account. The machine executing a task replication will communicate with the controller after the task replication generates a



▲**Figure 2. Non-overlapping data block transmitting.**

set of data records. The same task replications will generate great numbers of data records in big data processing. The communication between the controller and task replications greatly increases the network overhead. Let c_c be the cost caused by the communication.

To reduce the cost generated by huge communication, we make a rule to transmit the group of key-value pairs instead of transmitting one key-value pair. Similarly, the controller chooses the first finished the data block group to transmit and discards the data block group generated by other task replications. The challenge is to determine how many key-value pairs should be grouped together. How to divide the data block into groups will be discussed in section 5.

The delay, another crucial fact needed to take into consideration, arises from transmitting the data block group. The data block is not transmitted immediately when it was been generated. If too many key-value pairs are grouped, the data receiving at the next task may have a large delay that seriously slows down the data processing. Otherwise there is a big communication overhead with the controller, almost same with the scheme without grouping. Similarly, the delay causes cost. Let $\min\left(c_c+\sigma\sum_i c_d^i\right)$, be the cost caused by delay. In this paper, we try to make a good tradeoff between c_d and c_c.

Although the approach of combining task duplication and pipelined intermediate data can significantly accelerate data processing, redundant data are generated and may incur congestion on the network. In this paper, we design a novel online shuffling with a traffic controller to eliminate redundant data.

4 Model and Problem Formulation

The MapReduce programming model consists of two primitives: map phrase and reduce phrase. The former phrase produces a set of intermediate records (key/value pairs), which are provided as input to the reduce phrase. As a result, transmitting the intermediate data through the network generates traffic cost. The cost of delivering a certain amount of data over a network link is evaluated by the product of data size and distance between two machines.

Now we formulate the network traffic minimization problem. All symbols and variables used in this paper are shown in **Table 1**. We first consider the data delivering between two machines. Let c_{xy} denote the traffic cost of transmitting data from machine $x \in M$ to machine $y \in R$, which can be calculated by:

$$c_{xy}=v_x d_{xy},\quad x\in M,\quad y\in R. \tag{1}$$

The receiving delay cost of sending group i is c_d^i. To make a good tradeoff between control overhead and data receiving delay, we attempt to minimize the following function:

$$\min\left(c_c+\sigma\sum_i c_d^i\right), \tag{2}$$

▼Table 1. Symbols and variables

Notations	Description
M	A set of machines to generate intermediate data
R	A set of machines to receive intermediate data
v_x	Data volume of an intermediate record produced by machine x
c_{xy}	The traffic cost from machine x $\in M$ to machine y $\in R$
c_c	The cost caused by communication with the controller
c_t^i	The total cost of transmitting group i
c_d^i	The delay cost of group i
d_{xy}	The distance between two machines x and y

where σ is the competitive ratio indicating the relative weight of delay cost [14]. This problem is challenging because we have no knowledge of generation time of key-value pairs in future. Therefore we design an online grouping algorithm that groups key-value pairs in data buffer. The detailed design will be presented in next section.

5 Design of Online Distributed Algorithm

Our proposed online grouping algorithm makes a good tradeoff between control overhead and data receiving delay. In an online big data processing environment, the data is pipelined between operators. The intermediate data should be transmitted to next phrase immediately. In our proposed work, the duplications of a task have the same output, but the time to produce each intermediate data record is different. By using a controller, the task duplications can communicate with the controller to confirm whether it transmits or discards its output data.

Another issue we need to solve is that the large number of communication times between a task duplication and the controller will increase the network overhead. Therefore, we propose an online distributed algorithm (**Algorithm 1**) to achieve the transmitting group of data records.

Algorithm 1. Online Distributed Algorithm

```
1:  c_d ← 0, c_c ← 0
2:  while data processing not over do
3:    if confirmation message is received then
4:      remove and transmit confirmed packets from buffer
5:      begin buffering new packet
6:      c_d ← 0, c_c ← 0
7:    else if c_d + c_c ≥ c_t then
8:      send transmitting request
9:    else
10:     continue buffering packets
11:     recalculate c_d and c_c
12:     end if
13: end while
```

We consider that multiple copies are launched for a task, and their intermediate outputs are sent to the corresponding task. Since task copies get the same input data, they generate the same output in the forms of key-value pairs. At the beginning of Algorithm 1, we define the variables c_d and c_c to indicate the delay cost of the data records in buffer and the communication cost between task duplication and the controller, respectively. Note that the algorithm is executed on each task duplication. There is a buffer to store the temporary data records. If the task duplication receives the confirmed message from the controller (line 3), it will remove the buffered packets and transmit packets to next phrase (line 4). Otherwise, it will check the sum of c_d and c_c, and then compare the sum with c_t. If the sum is larger than c_t, it will send the transmitting request to the controller (line 8). We set the condition (line 7) to buffer the produced data sets into one group. The reason is that our primary goal is to minimize network traffic, and therefore the extra cost cannot be larger than the network traffic cost.

6 Evaluation

We conduct extensive simulations to evaluate the performance of our proposed work with the following two operations.

- Only the controller manages the execution of task duplications to deliver the intermediate key-value pairs without overlap.
- Each task duplication executes the proposed algorithm to reduce the network overhead. To compare with our work, the random allocation algorithm is executed on each machine to deliver group of data records.

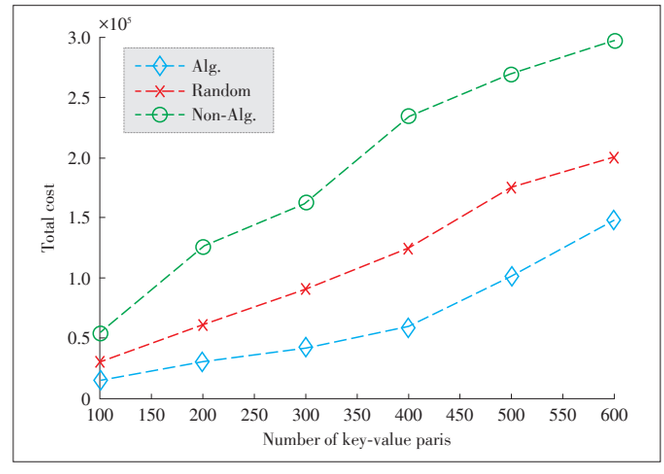
To our best knowledge, we propose to mitigate task stragglers and minimize total network cost for pipelining environment. The intermediate data records (*key/value pairs*) are associated with random size within [1−50], and each task has 30 task duplications in our simulations. The time of producing an intermediate data records is randomly set within [2−6] ms. For comparison, we also show the results of an random grouping algorithm and the one without traffic control.

We first evaluate the performance of our proposed algorithm by comparing with the random allocation algorithm and no group allocation algorithm. The controller chooses the machine that finishes its assigned task first, and then applies algorithms to divide the intermediate data into different groups. First of all, we tested intermediate key values' numbers of 100, 200, 300, 400, 500 and 600 for pipelining data processing output, and set the fixed task numbers to 30. As shown in **Fig. 3**, although the total cost increases as the number of key-value pairs increase for all the cases, our proposed algorithm has much lower increase than the other two schemes.

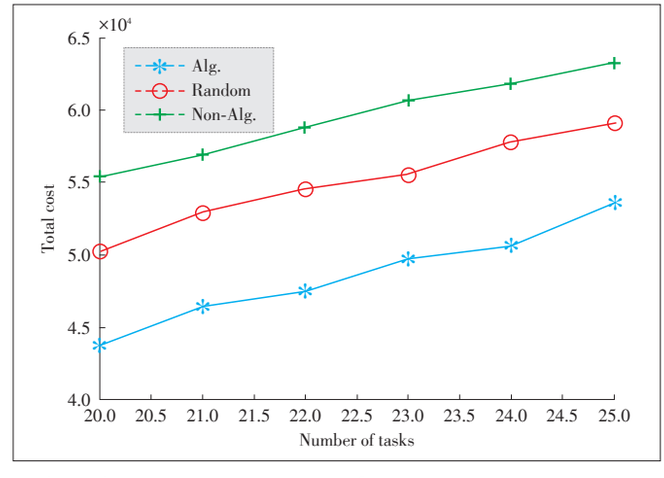
We then test the performance under the case of the fixed number of intermediate key value pairs, 300. As shown in **Fig. 4**, the total cost shows as an increasing function of number of tasks from 20 to 25 for all the cases. In particular, when the number of tasks is set to 25, the total cost of our proposed algorithm is about 5.3×10^4, while the total cost of no algorithm is 6.3×10^4, with a reduction of 25 percent. In contrast to the random allocation algorithm, our proposed algorithm also has a higher performance, because the random allocation algorithm does not consider the data delay.

As shown in **Fig. 5**, we study the influence of different number of task duplications by increasing the number of duplications form 10 to 15. The results show that our proposed algorithm always outperforms the other two schemes. That is because it requires only 2 times communication with the controller after delivering a group of data records, which is much smaller than the communication times without algorithm. Moreover, we make a tradeoff between the communication cost and the data delay cost and try to minimize the total cost. However, the random allocation algorithm does not take the data delay cost into consideration. Therefore, the total cost of the random allocation algorithm is much bigger than our proposed algorithm in the same cases.

Finally, we study the performance of three schemes under



▲Figure 3. Total cost vs. the number of key-value pairs.

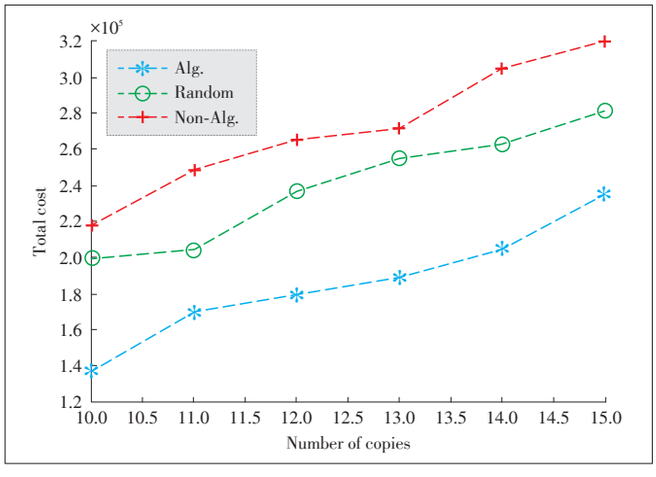


▲Figure 4. Total cost vs. the number of tasks.

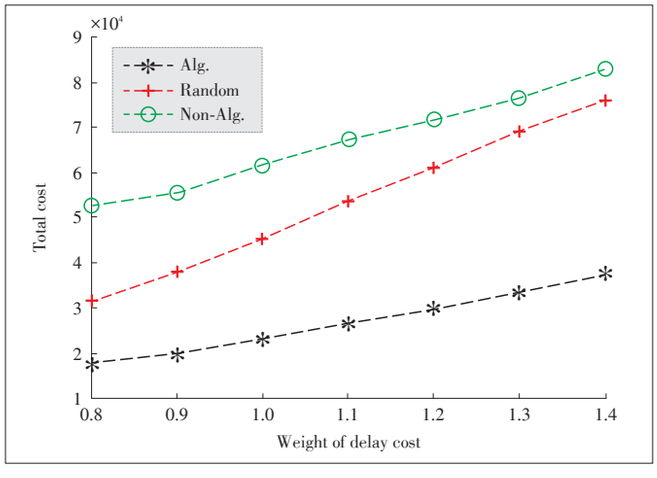
different weights of ratio by changing its value from 0.8 to 1.4. A small value of ratio indicates less importance of delay cost. From **Fig. 6**, we observe that the total cost of the random allocation algorithm has a sharp increase. However, the curve of our proposed algorithm increases slowly and it still outperforms the other two schemes.

7 Conclusions

In this paper, we study the principle of the parallel computing frameworks in cloud. We also use the technique of task duplication and strive for a traffic awareness for online big data processing. This approach can effectively accelerate job execution while avoiding redundant data transmission. To make a good tradeoff between network overhead and data delay, we design an online grouping algorithm to eliminate the traffic cost in cloud network. Finally, the performance of the proposed algorithm is evaluated by extensive simulations.



▲**Figure 5. Total cost vs. the number of task duplications.**



▲**Figure 6. Total cost vs. weight of delay.**

References

[1] S. Zou, X. Wen, K. Chen, et al., "Virtualknotter: online virtual machine shuffling for congestion resolving in virtualized datacenter," *Computer Networks*, vol. 67, pp. 141–153, 2014. doi: 10.1109/ICDCS.2012.25.

[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: attack of the clones," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI' 13)*, Lombard, USA, 2013, pp. 185–198.

[3] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, et al., "Grass: trimming stragglers in approximation analytics," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI' 14)*, Seattle, USA, 2014, pp. 289–302.

[4] C.-Y Hong, S. Kandula, R. Mahajan, et al., "Achieving high utilization with software-driven WAN," in *SIGCOMM' 13*, Hong Kong, China, 2013, pp. 15–26. doi: 10.1145/2534169.2486012.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proc. 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 04)*, San Francisco, USA, 2004, vol. 6, pp. 10–10.

[6] The Apache Software Foundation. (2017, Mar. 29). Apache Hadoop [Online]. Available: http://hadoop.apache.org

[7] T. Condie, N. Conway, P. Alvaro, et al., "Mapreduce online," in *Proc.7th USENIX Conference on Networked Systems Design and Implementation (NSDI' 10)*, San Jose, USA, 2010, pp. 21–21.

[8] G. D. Ghare and S. T. Leutenegger, "Improving speed up and response times by replicating parallel programs on a snow," in *11th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Cambridge, USA, 2005, pp. 264-287.

[9] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 08)*, San Diego, USA, 2008, pp. 29–42.

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Lisbon, Portugal, 2007, pp. 59–72. doi: 10.1145/1272996.1273005.

[11] G. Ananthanarayanan, S. Kandula, A. Greenberg, et al., "Reining in the outliers in map- reduce clusters using mantri," in *Proc. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 10)*, Vancouver, Canada, 2010, pp. 265–278.

[12] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wijckoff, "Char- lotte: metacomputing on the Web," *Future Generation Computer Systems*, vol. 15, no. 5–6, pp. 559–570, 1999.

[13] M. C. Rinard and P. C. Diniz, "Commutativity analysis: A new analysis framework for parallelizing compilers," in *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Philadelphia, USA, 1996, pp. 54–67.

[14] Q. Zang, H. Y. Chan, P. Li, and S. Guo, "Software-defined data shuffling for big data jobs with task duplication," in *45th International Conference on Parallel Processing Workshops (ICPPW)*, Philadelphia, USA, 2016, pp. 403– 407. doi: 10.1109/ICPPW.2016.62.

Manuscript received: 2017-06-23

Biographies

ZANG Qimeng (zangqm.uoa@gmail.com) is a graduate student in the department of Computer Science and Engineering, The University of Aizu, Japan. His research interests mainly include big data, cloud computing and RFID system.

GUO Song (song.guo@polyu.edu.hk) received his Ph.D. in computer science from University of Ottawa, Canada. He is currently a full professor at Department of Computing, The Hong Kong Polytechnic University (PolyU), China. Prior to joining PolyU, he was a full professor with The University of Aizu, Japan. His research interests are mainly in the areas of cloud and green computing, big data, wireless networks, and cyber-physical systems. He has published over 300 conference and journal papers in these areas and received multiple best paper awards from IEEE/ACM conferences. His research has been sponsored by JSPS, JST, MIC, NSF, NSFC, and industrial companies. Dr. GUO has served as an editor of several journals, including *IEEE TPDS*, *IEEE TETC*, *IEEE TGCN*, *IEEE Communications Magazine*, and *Wireless Networks*. He has been actively participating in international conferences serving as general chairs and TPC chairs. He is a senior member of IEEE, a senior member of ACM, and an IEEE Communications Society Distinguished Lecturer.