International Journal of Innovative Computing, Information and Control
Volume **16**, Number **3**, June **2020**

ICIC International ©2020 ISSN 1349-4198
pp. 1021–1033

DYNAMIC PROGRAMMING ALGORITHM AND BAT ALGORITHM BASED STORM NODES SCHEDULING IN EDGE COMPUTING

Lingling Zhang

Department of Computer Science and Technology
Beihua University
No. 3999, Binjiang East Road, Jilin 132013, P. R. China
373752782@qq.com

Received October 2019; revised February 2020

Abstract. *The high communication delay and uneven load among heterogeneous edge nodes are affecting the performance of edge computing, and they are almost impossible to be solved by the traditional cloud computing platforms. In this paper, we address these problems by studying the scheduling optimization method for the storm nodes in edge computing environments. At first, a storm scheduling model is established, where server cluster structure and schedule workflow are formulated. Then, a heuristic dynamic programming algorithm is proposed to address the general scheduling issue and a bat-based scheduling strategy is proposed to address a special case faced by the heuristic dynamic programming algorithm. Finally, the experimental results show that the proposed algorithm can minimize the communication cost and guarantee the minimum scheduling requirements.*

Keywords: Edge computing, Storm nodes, Scheduling, Dynamic programming, Bat algorithm

1. Introduction.

A lot of edge devices are used to pre-process tremendous amount of data in edge computing for the purpose of unloading computing tasks to multiple edge nodes, reducing bandwidth consumption and data transmission difficulty, etc. Due to such operations, Quality of Service (QoS) is greatly improved [1]. In addition, the role of edge computing is particularly important for the network scenarios which have the extremely high real-time requirements.

Generally, Internet of Things (IoT) intends to unload the heavy computing tasks to edge servers in order to reduce the energy consumption caused by the centralized computing [2,3]. However, it is aware that the operations of unloading tasks will also consume the additional energy due to transmitting tasks to the edge servers [4], which in turn increases the overall completion time of unloading tasks. In this way, the task scheduling problem in edge nodes has become a hot topic [5-7]. In fact, the scheduling method is the key for solving the task scheduling problem, because the appropriate method can i) minimize the scheduling time among edge nodes, ii) minimize the communication cost and make the most use of node resources, and iii) guarantee the network load balance in server clusters. However, this kind of problem has already been proved as NP-hard, which means that it cannot be addressed within the polynomial time [8].

There are two main kinds of cloud computing platforms, i.e., batch-processing ones and streaming processing ones. Among them, it is noted that only the platforms (e.g., Apache Storm [9]) having high real-time streaming processing features can meet the application requirements of edge computing. Despite this, the EvenScheduler adopts the

DOI: 10.24507/ijicic.16.03.1021

polling method to allocate tasks, which may cause high communication cost, unbalanced network load condition, and even low edge node resource utilization. All these situations would seriously affect the transmission performance of edge computing. The lastest ResourceAwareScheduler adopts a new scheduling strategy, which optimizes communication cost and resource utilization by using scheduler to sense the status of node resources, such as performance, energy and thermal [10]. However, it is unnecessary to change the allocation mode of the Storm's topological task threads, since it only requires to make the resource awareness, to optimize the node resource utilization and the edge node load balance. Hence, the scheduling time is still long and network load may still be uneven in the edge computing environment, that is to say, the currently existing Storm scheduling mechanisms cannot meet the requirements of edge computing [11].

In order to address the above problems, this paper studies the global scheduling optimization method of Storm nodes in edge computing. First of all, we establish a Storm scheduling model for the edge computing, in which the server cluster structure and schedule workflow are carefully explained and formulated. Targeting on this, a heuristic dynamic programming algorithm is proposed to address the general scheduling issue. Then, a bat-based scheduling strategy is proposed to address a special case facing the heuristic dynamic programming algorithm. Anyway, the two methods are designed and studied in order to present a suitable scheduling optimization method for the Storm.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents the scheduling and optimization model, while Section 4 gives the corresponding algorithm design. The experiments results are discussed in Section 5 and Section 6 concludes this paper.

2. **Related Work.**

Currently, there are already a lot of works studying the edge computing. However, most of them are transformed from the study of IoT, which may cause the unexpected issues, since the exact characteristics are different from those of edge computing. In this section, we focus on exploring the works that deeply study into the areas of edge computing.

First of all, an important challenge to the scheduling problem in edge computing optimization and the real-time streaming processing framework is how to optimize the deployment of the so-called network task topologies [12]. With respect to this, the recent researches have expanded their Storm scheduling work from the isomorphic environment to the heterogeneous scenario. More specifically, some of them have set up the additional inputs which include link information and resource utilization, and the additional modules which include system monitors and even complex schedulers [13,14]. By introducing these additional elements, the Storm scheduling can be flexibly configured as needed. For example, the achievements in [15-18] are based on an extension of the original Storm by taking account of CPU and network load to establish a rebalanced task-to-node allocation. Besides, [19] relies on the Metis tool to divide the task topology into K layers, while [20] presents a method which relies on suing GPU to improve the computing efficiency of Storm.

However, these works still suffer from many challenges. For instance, although [15] uses the additional resource-aware detection module to optimize communication cost and resource utilization, it inevitably increases the operation complexity of scheduling process. In addition, [17] relies on using the greedy algorithm to detect network delay, while it easily falls into a local optimum which may cause the unexpected situations [21]. Overall, most of the above works address the Storm scheduling issue by the additional auxiliary modules without essentially changing the topology distribution model [18]. In this way, they still suffer from the problems of long scheduling time and uneven network load. Hence, how

to cut down scheduling time and communication cost, and how to improve load balance and resource utilization are the urgent issues to be solved.

At present, there are also some related research works that use Dynamic Programming Algorithm (DPA) [22] to optimize such scheduling issue in edge computing. For example, [23] proposes a heuristic DPA to change the arrangement and allocation of tasks in the Storm scheduling framework. In addition, [24] proposes to check such mapping relationship periodically, and it calculates the optimal scheduling scheme based on the detected information (i.e., configuration of edge nodes and global network topology information). Such heuristic strategies can accurately calculate the global optimal scheduling within the range of concurrency degree set by Jave Virtual Machine (JVM) [25]. In particular, these references also have some disadvantages. For example, if the concurrency degree of the topological task is higher than the default threshold, the stack overflow problem will be caused and the unexpected trouble will appear [26].

With respect to the research topic of task scheduling, many intelligent algorithms are adopted, such as particle swarm optimization, ant colony optimization, and genetic algorithm [27-30]. Due to the complexity of task scheduling, it is an NP-hard problem, and these intelligent algorithms can actually address such NP-hard problem within the polynomial execution time and achieve a near-optimum result [31]. Despite the fact that these intelligent algorithms have already been widely used in the field of network science and engineering, their inner-robustness and their abilities to deal with the problems having different characteristics and attributes are still not very satisfactory [32]. Compared with the above intelligent algorithms, Bat Algorithm (BA) has appeared as one more efficient method. Specifically, it can deal with some problems more effective than genetic algorithm and particle swarm optimization method [33]. Based on the characteristic of BA, we can easily find the scheduling timing. Therefore, in this paper, we propose a task scheduling strategy based on BA that first initializes a random solution based on the mapping relationship between task instance and slots for the Storm scheduling problem. Then, we calculate the optimal results by continuously and iteratively executing BA.

3. Scheduling Model and Optimization Scheme.

3.1. Cluster architecture.

The Storm cluster in edge computing is composed of one master cell node in the Storm cloud computing center and many heterogeneous edge servers in the edge cloud, as shown in Figure 1.

In particular, the Storm edge computing master unit is the communication master unit node between cloud computing center and edge cloud. In other words, it manages the processes of communication and scheduling when unloading the cloud center topology flow tasks to the edge server nodes in the Storm cluster. Taking the case in Figure 1 as the example, it loads the Nimbus as the core component of Storm, which is in charge of collecting and managing the topological data stream uploaded from the cloud computing center to the edge servers. Then, these data stream will be allocated to certain sub-nodes of each edge server according to the corresponding scheduling algorithm.

Usually, the child node will load the Supervisor components and such Supervisor will have one or more Worker processes. Each Supervisor is responsible for delegating tasks to the Worker processes, such that the Worker processes will need to generate as many Executor threads as possible in order to run the large number of topological tasks. Moreover, all sub-nodes that run in the edge servers will naturally form an edge cloud environment.

3.2. Scheduling model.

For each node in the Storm cluster, its Supervisor will start one or more worker processes. All the topologies have to run in the worker processes. In particular, the maximum number of processes that can be started is determined by the

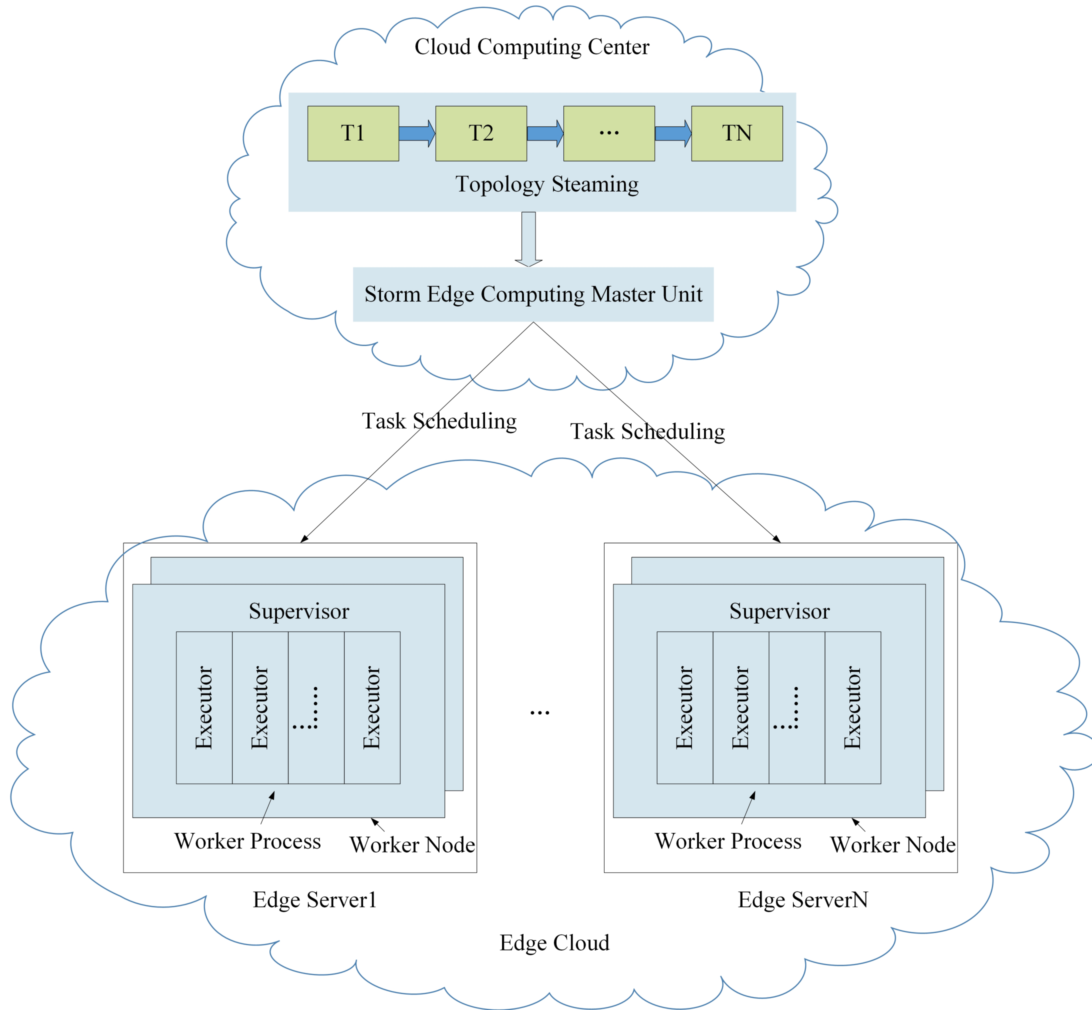

FIGURE 1. Storm's cluster architecture

Slot configured by the corresponding node. Here it is noted that the Slot actually refers to the number of ports. Generally, the default Slot in the Apache Storm is bounded varying from 6700 to 6703. One single worker process can run multiple Executor threads, while each Executor thread can run only one single task instance or execute one component.

The existing Storm scheduling models usually rely on using the sort-slot algorithms to periodically allocate the task instances of Executor threads to the available slots one by one in the form of <node id+port id, (start-task-id, end-task-id)>. Such method follows a very simple sorting and redistributing pattern, which may result in unbalanced load among edge nodes, especially when the number of topological tasks becomes large. In addition, it should also be noted that the configuration information of nodes has not yet been taken into consideration.

In this paper, we intend to change the scheduling model. In particular, the updated model consists of a set of task instances allocated to Executor threads by each slot, where such set form can be expressed as a one-dimensional array. At the same time, we also add an additional module to the proposed model for the purpose of obtaining the configuration information of edge nodes. The corresponding scheduling model is shown in Figure 2.

Let N denote a Storm cluster, and the structure of the Storm cluster is defined as follows. For any one Storm cluster having n work nodes, we have $N = \{n_i | i \in [1, n]\}$, where each n_i is configured with S_j slots. Furthermore, let R denote the computing

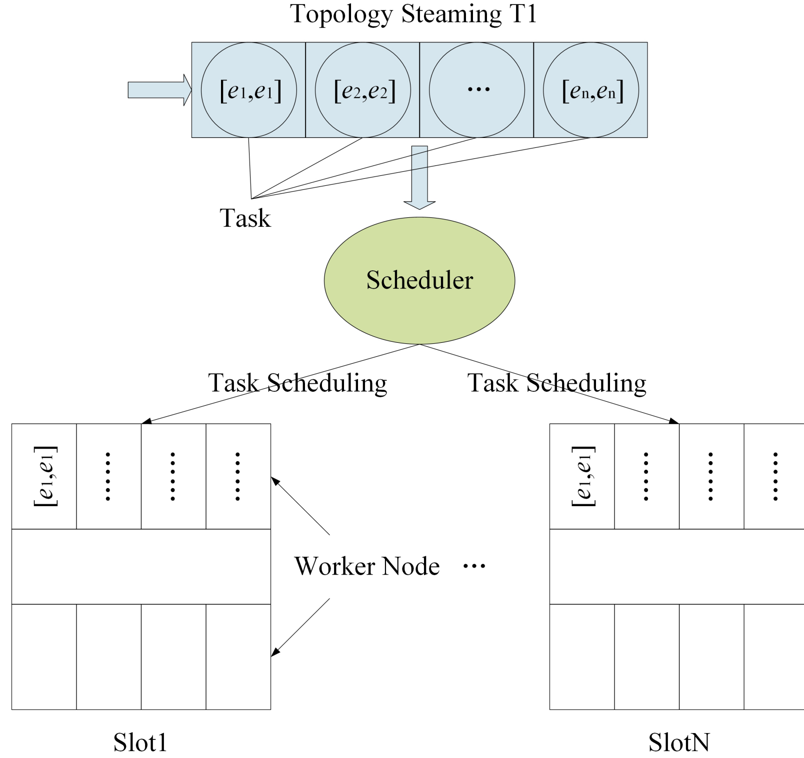

FIGURE 2. Storm's task scheduling model

resources set of cluster that can be allocated, and we have

$$R = \left\{ S_j^i =< i, j > | i \in [1, n],\ j \in [1, S_j] \right\} \tag{1}$$

where S_j^i is the j-th slot of n_i.

For the topology T to be submitted to the cluster, the structure of a task's T consists of the Executor threads. Besides, the task instance (e.g., spout or bolt) in Executor is a sequence of two-dimensional array consisting of begin task id and end task id, and the corresponding unit format can be defined as follows.

$$E_i = [start_{id}, end_{id}], \quad i \in (1, N) \tag{2}$$

where $start_{id}$ and end_{id} of a task instance are usually the same, that is, $start_{id} = end_{id}$. The scheduler in Figure 2 allocates task instances of the topological data flow T1 to the corresponding edge nodes in the form that a set of number of task instances allocated to each slot in the Executor thread.

For each task instance in T, the $N_e(T)$ executors will be distributed in the set form of $[start_{id}, end_{id}]$ to the corresponding Slot set of nodes. On this basis, the Executor threads are stored in each Slot collection, which corresponds to the number of Slots that have been allocated. Then, the resource scheduling for T can be expressed as follows.

$$f(x) \to S \tag{3}$$

where the function f indicates the mapping from Executor to Slot and S indicates the corresponding Slot. Meanwhile, the resource scheduling should satisfy the following two requirements: (i) the number of workers occupied by T should be smaller than or equal to the number of Slots in the cluster; (ii) two executors in different topologies are not allowed to be allocated to the same worker.

3.3. **Optimization method.** In this paper, we simplify the Storm scheduling problem as how to allocate $N_e(T)$ Executor threads to $N_S(T)$ Slot sets, while minimizing scheduling time in Storm edge nodes, maximizing resource utilization and guaranteeing load balance among nodes. In order to optimize the above scheduling process, we first regard the allocation of $N_e(T)$ Executor to $N_S(T)$ Slots as one solution for the cluster. Then, the number of solutions is determined by the number of Executor threads and the number of Worker processes. The structure of each solution is actually a one-dimensional array which is composed of the Executors allocated to S_j Slot. In particular, the number of Slots (i.e., $N_S(T)$) and the number of Executors (i.e., $N_e(T)$) in T are the main objects to be optimized.

Besides, the configuration information of edge nodes obtained by the additional configuration detection module is used as the input of scheduling. The total execution time of a task scheduling and the standard deviation of load balance of each edge node are used as the evaluation value of the solution. The optimal solution is calculated by using heuristic DPA and BA based scheduling strategy.

4. The Proposed Algorithm Based on DPA and BA.

4.1. **Heuristic DPA.** In this paper, we consider the detection results of edge node configuration (i.e., CPU utilization) as the fitness function to evaluate the obtained solution. Let res_i denote the solution generated by the i-th allocation scheme, C_{sys} denote the CPU resource allocated to the cluster, and r_{exe} denote the percentage of allocating CPU resource to the Executors, and we have the execution time of Slot as follows.

$$T_i = \sum_{i=1}^{N_S(T)} \frac{res_i^2 \cdot r_{exe}}{C_{sys}} \tag{4}$$

where T_i is the execution time of the i-th Slot and the small T_i means the short overall execution time of the task. When calculating the shortest execution time of each solution, it is necessary to consider load balance of each node, that is, the time spent by each node to perform tasks. Meanwhile, the small fluctuation range means good load balance. On this basis, let LB denote the standard variance on load balance, and we have

$$LB = \sqrt{\frac{1}{N_S(T)} \sum_{i=1}^{N_S(T)} (T_i - T_{avg})^2} \tag{5}$$

where T_{avg} is the average with respect to $N_S(T)$ execution times.

As the above mentioned, the purpose of this algorithm is to assign $N_e(T)$ Executors to $N_S(T)$ Slots, during which the overall execution time is guaranteed to be the minimum and the load balance degree is the maximum. Given this, the proposed algorithm should first initialize the solution set as $res = \{res_1, res_2, \ldots, res_n\}$. Let id_i denote the index of the i-th Slot, $MaN_e(T)$ denote the maximal number of Executors that can be contained by each Slot and $MiN_e(T)$ denote the minimal number of Executors that can be contained by each Slot, and the process of algorithm is described as follows.

Step 1. Initialize the input configurations of T, which include $N_e(T)$, $N_S(T)$, $MaN_e(T)$ and $MiN_e(T)$.

Step 2. Initialize the currently allocated number of Executors as zero, i.e., $N_e(T) = 0$;

Step 3. For the i-th Slot (i.e., the index is id_i), if the allocated number of Executors is smaller than $N_e(T)$, then the values within the boundary of $MaN_e(T)$ and $MiN_e(T)$ are checked until the id_i-th position of res_i is satisfied.

Step 4. Recursively put the unallocated Executors into res_i until id_i reaches $N_S(T)$.

4.2. **BA based scheduling strategy.** For the scheduling strategy, it is designed based on BA. Firstly, this strategy initializes a group solution randomly according to the workflow in Figure 2. Then, it iteratively calculates the global optimal solution according to Equations (4) and (5). After that, the Executor will be allocated to the Storm cluster in the form of $< start_{id}, end_{id} >$ according to the number of optimal solutions. Finally, in order to optimize high communication cost problem, the Executor threads belonging to the same Slot will be grouped together. The corresponding process based BA is described as follows.

Step 1. Traverse T to determine whether the topology needs to be scheduled: if yes, the following steps are performed continuously; otherwise, the algorithm is finished.

Step 2. Get the map set that illustrates the mapping between the component ids to the executor ids for T. The map set that hosts the component ids is stored in the new set, and the executor ids will be sorted according to the order of the component ids and finally stored in the connection set.

Step 3. Determine the number of Slots needed according to the number of workers configured by T.

Step 4. Based on the solution by BA, allocate the Executors to the workers according to the allocation scheme, and finally store such results in the list set.

Step 5. Obtain and sort the Slots available to the cluster and then store them into the list collection. If the Slots are full, the release operation is performed.

Step 6. Invoke the proposed allocation method to allocate the Slots to the edge nodes of cluster, and end the algorithm.

5. Performance Evaluation.

5.1. **Setup.** The experiments are carried out on a Dell R710 server with the ESXI 6.0 system, in which a virtual cluster is established and used to simulate the interaction among the edge servers. Four different virtual nodes are also simulated with different configurations. The basic configuration of the server is as follows. CPU: Intel (R) Xeon (R) X5650, 2.66 GHz * 6 core * 2, RAM: 128GB, 1 Gbps * 4 network card and 2 * 1T hard disks. The configurations of four virtual nodes are as follows: (i) 2.67 GHz * 1 CPU core, 2GB RAM and the Ubuntu 16.04 X86-64 operating system, (ii) 2.67 GHz * 2 CPU core, 4GB RAM and the Ubuntu 14.04 X86-64 operating system, (iii) 2.67 GHz * 4 CPU core, 6GB RAM and the Ubuntu 14.10 X86-64 operating system, and (iv) 2.67 GHz * 6 CPU core, 8GB RAM and the Ubuntu 12.04 X86-64 operating system. In addition, the cluster configuration is shown in Table 1.

Table 1. Storm nodes configuration

Host	IP address	Function
Storm-M	192.168.0.15	Nimbus, Zookeeper
Storm-S1	192.168.0.16	Supervisor, Zookeeper
Storm-S2	192.168.0.17	Supervisor, Zookeeper
Storm-S3	192.168.0.18	Supervisor, Zookeeper

In this paper, the lastest ResourceAwareScheduler is used to implement the proposed algorithm. In addition, the Word Count topology is used to test the cluster performance. In order to ensure the accuracy of time-related measurement data, the time in the cluster is synchronized, that is, all nodes in the cluster are configured with the time synchronization protocol.

5.2. **Metrics.** High real-time processing requirement in edge environment is the key for the Storm edge node scheduling optimization problem. Besides, the throughput of cluster is another key indicator affecting the real-time processing performance of cluster, which refers to the amount of data that can be processed per time unit. Particularly, the CPU utilization and load balance are the main factors determining the cluster's throughput. In this way, the following indicators are used for performance evaluation: (i) the throughput of Tuple per time unit time, which means the number of the Tuples that can be processed per second, which reflects the throughput of the cluster; (ii) the average CPU utilization among all edge nodes in the cluster within a specified time; and (iii) the standard deviation of average CPU utilization for all edge nodes in the cluster within a specified time.

5.3. **Benchmarks.** The experimental data is obtained from (i) the Trident RAS API, (ii) the Storm UI REST API, and (iii) the edge node configuration detection module implemented by this algorithm. Based on these data, two benchmarks are used for comparison, i.e., (i) Storm which is the default balanced scheduling model of Storm and (ii) R-Storm which is the resource aware and real-time Storm model proposed in [10]. In particular, the edge-oriented Storm model proposed and implemented in this paper is called E-Storm.

5.4. **Results.**

5.4.1. *CPU utilization.* Table 2 shows the average CPU usage rate of all edge nodes. The corresponding results are measured every 5 seconds within 1 minute after the beginning of the topological task in the scheduling scenario of the three scheduling models. The unit of the value in Table 2 actually means the percentage of CPU usage in the topological task processing. In order to show the optimization comparison results of these three scheduling models in a more intuitive way, this paper presents the relationship between CPU occupancy rate and elapsed time as shown in Figure 3. Apparently, we can discover that the performance of E-Storm is always better than that of Storm, and partially better than that of R-Storm. Despite the fact that the overall CPU occupancy rate of R-Storm is slightly higher than that of E-Storm, we can tell that the gap between them is not very obvious.

Table 2. The average CPU usage rate within 1 minute

Timestamp (s)/CPU usage rate (%)	Storm	R-Storm	E-Storm
1	4.55	10.85	9.4
5	4.3	11.55	7.3
10	4.75	10.15	6.7
15	4.7	7.7	5.5
20	3.9	8.45	8.3
25	4.5	7.1	7
30	4.55	8.2	6.45
35	4.25	6.55	10.35
40	4.15	7.7	5.4
45	3.8	10.7	6.2
50	5.4	7.6	6.3
55	4.75	5.75	6.55
60	4.5	6.9	7.55

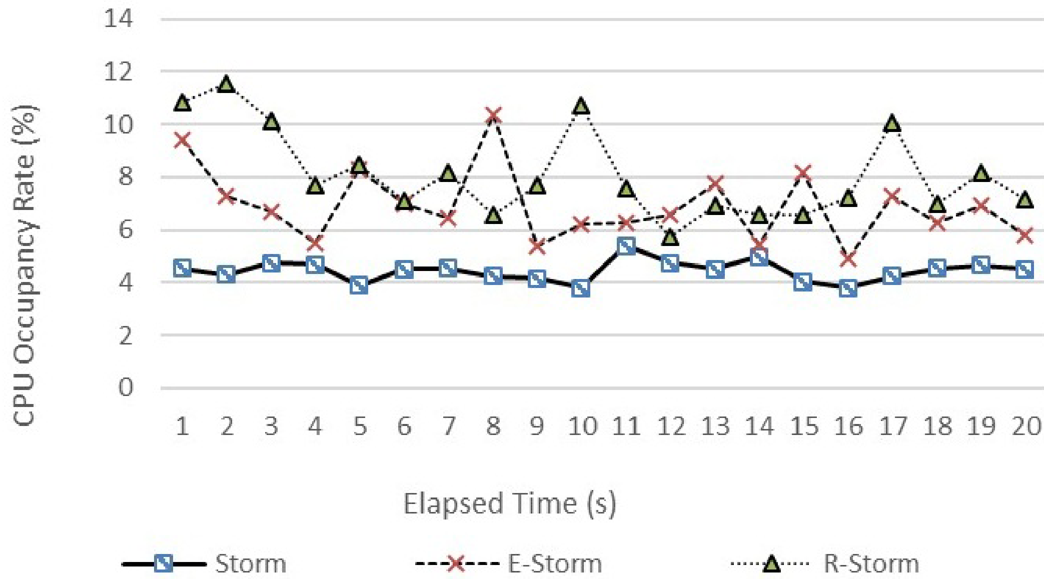

FIGURE 3. Average CPU occupancy rate

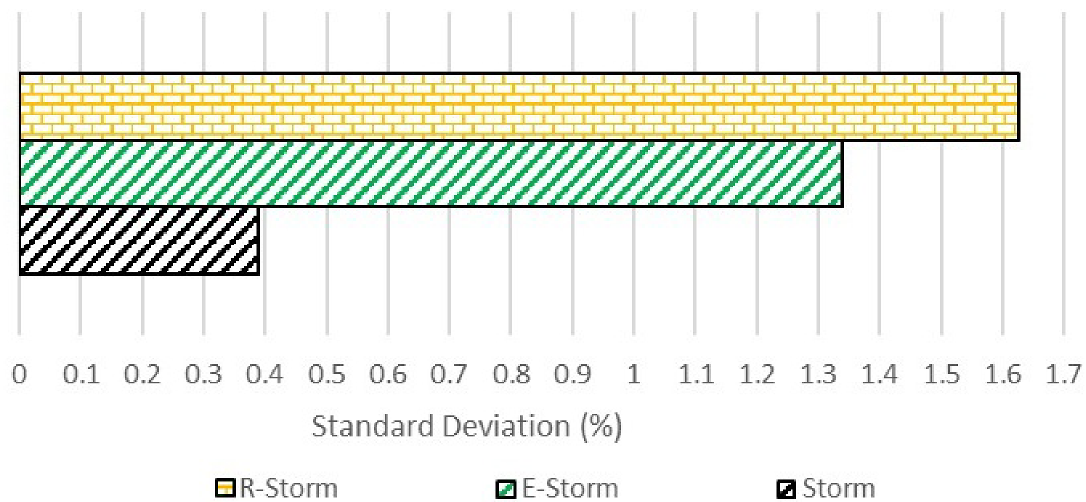

FIGURE 4. The standard deviation on average CPU occupancy rate

5.4.2. *Load balance.* In order to explain the load balance among edge nodes, the standard deviation is introduced. In particular, the standard deviation results of CPU usage for each node in the scheduling scenarios are illustrated in Figure 4. The results are achieved under three different scheduling models and different scenarios as explained in the previous section. Generally, the smaller the standard deviation is, the more balance will be. As can be discovered from Figure 4, the load balance of Storm is the best among them in the current experimental environment. Meanwhile, the load balance condition of E-Storm is obviously better than that of R-Storm, that is, R-Storm cannot effectively handle the load balance among edge nodes when fulfilling the scheduling optimization.

5.4.3. *Throughput.* The cluster throughput is actually a comprehensive indicator affecting the real-time data processing in edge computing. Generally, the larger the cluster

throughput is, the stronger the data processing capability will be per time unit. In this way, we summarize the corresponding results and show them in Table 3. For each scheduling model, we collect the throughput information of the cluster every 5 seconds within 1 minute from the beginning of the scheduling process. The unit for the values in Table 3 is in fact evaluated by the number of tuples processed per second. In order to show the difference of the throughput achieved in different clusters by different methods more intuitively, we compare the cluster throughput under each scheduler and obtain the corresponding results every 5 seconds within 1 minute after the beginning of the scheduling process. All the collected corresponding results are shown in Table 3 and Figure 5.

Table 3. The cluster throughput under each scheduler within 1 minute

Timestamp (s)/Throughput (Tuple/s)	Storm	R-Storm	E-Storm
5	220	600	420
10	1060	1420	1520
15	2200	2300	2420
20	3100	3480	3380
25	4040	4300	4540
30	5000	5380	5380
35	5980	6360	6380
40	6880	7240	7380
45	7880	8400	8240
50	8760	9260	9420
55	9700	10100	10180
60	10780	11240	12240
65	12780	12100	13120
70	13080	13800	14160

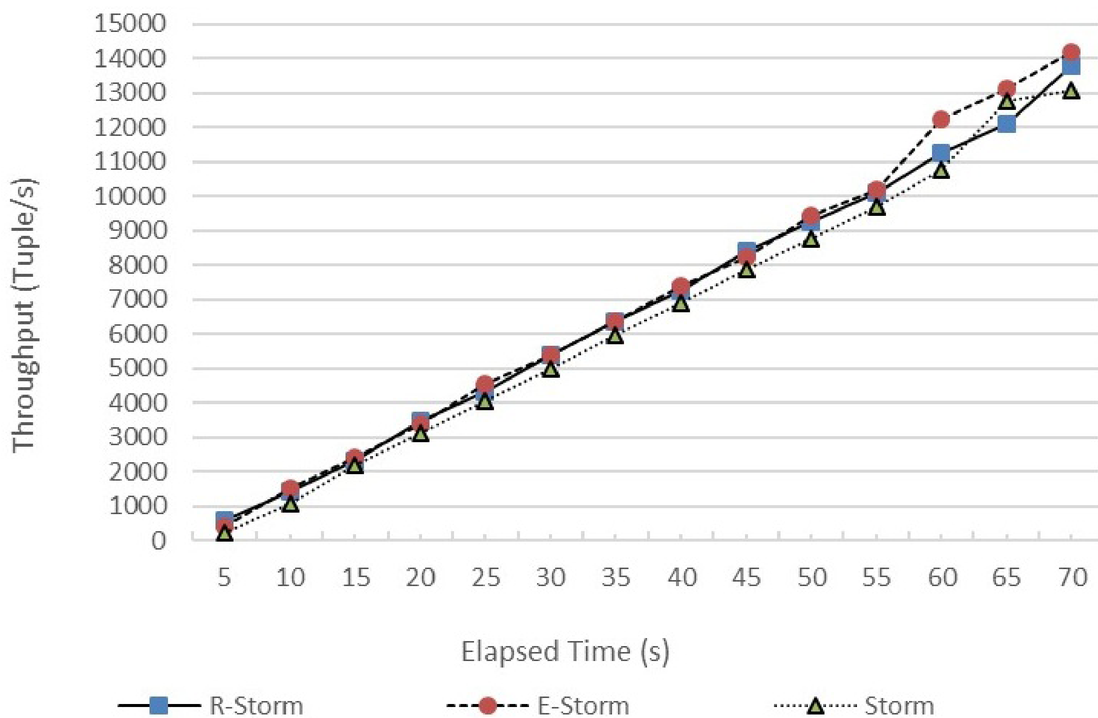

Figure 5. The tuple throughput per time unit

As can be discovered from Figure 5, the Tuple processing capacity of E-Storm is almost similar to that of R-Storm. However, we can discover that such situation only exists before the elapsed time of around 55 seconds. Specifically, after the elapsed time of 55 seconds, the throughput performance of E-Storm is obviously better than that of the R-Storm, which can reflect the efficiency of the proposed method in a certain extent. On the other hand, it is easy to note that the performance of Storm is significantly worse than that of E-Storm during all the elapsed times. Overall, we divide the whole period into two parts by the time of 55 seconds. For the first part, E-Storm does not show many benefits than that of R-Storm, while in the second part E-Storm achieves higher throughput than that of the other two models until the system finally stabilizes.

With respect to the comparison of the CPU usage and the load balance indicators, E-Storm does not have a good load balance as Storm does, while it actually achieves a very higher CPU utilization than that of Storm. This will eventually affect the throughput performance of clusters. Jointly taking the two indicators into consideration, E-Storm model proposed in this paper achieves better performance than the two methods with respect to the overall optimization of resource utilization and load balance.

6. **Conclusions.** This paper studies the computational unloading strategy of Storm edge nodes in the edge computing, and proposes a heuristic dynamic programming algorithm which can calculate all the allocation schemes with the requirements satisfied and find the global optimal solution accurately. To solve the problem that the concurrency of topological tasks may exceed the maximum stack depth set by JVM, the BA-based scheduling strategy is also proposed, which calculates the optimal solution iteratively through the random initial solutions. The proposed method can be applied to the most common scenarios without the need to configure parameters manually. The experiment results reveal that the proposed method can achieve better performance on the cluster throughput optimization. It can also meet the high real-time processing requirements among edge nodes and effectively improve the data transmission ability when addressing the scheduling optimization problems in the edge environment.

However, as a novel optimization strategy, there are also some limitations. For example, on the one hand, the combination of DPA and BA increases the computation complexity and the optimal solution is obtained by the multiple iterations, which affect the overall network performance. On the other hand, the proposed algorithm is not suitable to the large-scale edge computing environment. Given this, the future research will focus on the above two points. In addition, the collaboration among different works can greatly improve the performance of edge computing, which will be also studied.

Acknowledgements. This paper is supported by the Scientific and Technological Research Project from Education Department of Jilin Province (Grant No. JLZX20562019072 4110543).

REFERENCES

[1] S. Moller, F. Koster and B. Weiss, Modelling speech service quality: From conversational phases to communication quality and service quality, *The 9th International Conference on Quality of Multimedia Experience (QoMEX)*, Erfurt, pp.1-3, 2017.

[2] H. Guo, J. Liu, J. Zhang et al., Mobile-edge computation offloading for ultradense IoT networks, *IEEE Internet of Things Journal*, vol.5, no.6, pp.4977-4988, 2018.

[3] Q. Pham, L. B. Le, S. Chung et al., Mobile edge computing with wireless backhaul: Joint task offloading and resource allocation, *IEEE Access*, vol.7, pp.16444-16459, 2019.

[4] Y. Zhang, X. Chen, Y. Chen et al., Cost efficient scheduling for delay-sensitive tasks in edge computing system, *Proc. of 2018 IEEE International Conference*, pp.73-80, 2018.

[5] Y. Kim, J. Kwak and S. Chong, Dual-side optimization for cost-delay tradeoff in mobile edge computing, *IEEE Trans. Vehicular Technology*, vol.67, no.2, pp.1765-1781, 2018.
[6] D. Zeng, L. Gu, S. Guo et al., Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system, *IEEE Trans. Computers*, vol.65, no.12, pp.3702-3712, 2016.
[7] L. Gu, D. Zeng, S. Guo et al., Cost efficient resource management in fog computing supported medical cyber-physical system, *IEEE Trans. Emerging Topics in Computing*, vol.5, no.1, pp.108-119, 2017.
[8] C. Jian, J. Chen, J. Ping et al., An improved chaotic bat swarm scheduling learning model on edge computing, *IEEE Access*, vol.7, no.1, pp.58602-58610, 2019.
[9] D. Xiang et al., RB-Storm: Resource balance scheduling in Apache Storm, *The 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, Hamamatsu, pp.419-423, 2017.
[10] S. Alsubaihi and J. Gaudiot, PETS: Performance, energy and thermal aware scheduler for job mapping with resource allocation in heterogeneous systems, *IEEE the 35th International Performance Computing and Communications Conference (IPCCC)*, Las Vegas, NV, pp.1-2, 2016.
[11] S. Liu et al., An adaptive online scheme for scheduling and resource enforcement in Storm, *IEEE/ACM Trans. Networking*, vol.5, no.1, pp.1-10, 2019.
[12] B. Cheng, Edge-computing-aware deployment of stream processing tasks based on topology-external information: Model, algorithms, and a Storm-based prototype, *IEEE International Congress on Big Data*, 2016.
[13] Y. Liu, S. Zhao and B. Cheng, Multi-sensor data fusion system based on Apache Storm, *IEEE International Conference on Computer and Communications (ICCC)*, Chengdu, China, pp.1094-1098, 2017.
[14] Z. Chen, N. Chen and J. Gong, Design and implementation of the real-time GIS data model and sensor web service platform for environmental big data management with the Apache Storm, *International Conference on Agro-Geoinformatics*, Istanbul, pp.32-35, 2015.
[15] B. Peng, M. Hosseini, Z. Hong et al., R-Storm: Resource-aware scheduling in Storm, *ACM Middleware Conference*, 2015.
[16] L. Aniello, R. Baldoni and L. Querzoni, Adaptive online scheduling in Storm, *Proc. of the 7th ACM International Conference on Distributed Event-Based Systems*, pp.207-218, 2013.
[17] V. Cardellini, V. Grassi, F. Presti et al., Distributed QoS-aware scheduling in Storm, *Proc. of the 9th ACM International Conference on Distributed Event-Based Systems*, pp.344-347, 2015.
[18] V. Gupta and R. Hewett, Unleashing the power of hashtags in tweet analytics with distributed framework on Apache Storm, *IEEE International Conference on Big Data*, Seattle, WA, USA, pp.4554-4558, 2018.
[19] L. Eskandari, Z. Huang and D. Eyers, P-scheduler: Adaptive hierarchical scheduling in Apache Storm, *Proc. of the Australasian Computer Science Week Multiconference*, 2016.
[20] Z. H. Chen, J. L. Xu, J. Tang et al., G-Storm: GPU-enabled high-throughput online data processing in Storm, *IEEE International Conference on Big Data*, pp.307-312, 2015.
[21] M. Naruse, K. Sekiguchi and K. Nonaka, Coverage control for multi-copter with avoidance of local optimum and collision using change of the distribution density map, *Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, Nara, Japan, pp.1116-1121, 2018.
[22] Y. Wang, T. Li and W. Su, Research on the bi-level programming of underground logistics project based on public-private partnership mode, *ICIC Express Letters*, vol.13, no.6, pp.505-511, 2019.
[23] W. Zhang, Y. Hu, H. He et al., Linear and dynamic programming algorithms for real-time task scheduling with task duplication, *The Journal of Supercomputing*, vol.75, no.2, pp.494-509, 2017.
[24] Z. Lei, Y. Sun, Y. Song et al., The quantitative analysis of equipment operational test credibility based on dynamic programming and 0-1 integer linear programming, *International Conference on Industrial Informatics*, Wuhan, China, pp.107-110, 2015.
[25] Z. Zhuang, C. Tran, H. Ramachandra et al., Eliminating OS-caused large JVM pauses for latency-sensitive java-based cloud platforms, *IEEE the 9th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, pp.694-701, 2016.
[26] C. Greco, T. Haden and K. Damevski, StackInTheFlow: Behavior-driven recommendation system for stack overflow posts, *IEEE/ACM the 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, Gothenburg, pp.5-8, 2018.
[27] Y. Xie, Y. Zhu, Y. Wang et al., A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloudCedge environment, *Future Generation Computer Systems*, vol.97, pp.361-378, 2019.

[28] Y. J. Moon, H. C. Yu, J. M. Gil et al., A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments, *Human-Centric Computing and Information Sciences*, vol.7, no.1, pp.2-8, 2017.

[29] Y. Li and S. Wang, An energy-aware edge server placement algorithm in mobile edge computing, *IEEE International Conference on Edge Computing (EDGE)*, San Francisco, CA, pp.66-73, 2018.

[30] X. Tong, L. Sun and T. Guan, Discussion on the optimization of assembly process for urban rail vehicle based on the lean intelligent manufacturing model, *Chinese Automation Congress (CAC)*, Jinan, China, pp.1081-1084, 2017.

[31] T. Demir and T. E. Tuncer, Robust optimum and near-optimum beamformers for decode-and-forward full-duplex multi-antenna relay with self-energy recycling, *IEEE Trans. Wireless Communications*, vol.18, no.3, pp.1566-1580, 2019.

[32] Y. Jiang, J. Li, X. Guo et al., Motion trajectory control of underground intelligent scraper based on particle swarm optimization, *Chinese Automation Congress (CAC)*, Jinan, China, pp.2287-2291, 2017.

[33] W. Kongkaew, Bat algorithm in discrete optimization: A review of recent applications, *Songklanakarin Journal of Science and Technology (SJST)*, vol.39, no.5, pp.641-650, 2017.