Hindawi
Security and Communication Networks
Volume 2021, Article ID 5541974, 10 pages
https://doi.org/10.1155/2021/5541974

Research Article

Task Priority-Based Cached-Data Prefetching and Eviction Mechanisms for Performance Optimization of Edge Computing Clusters

Ihsan Ullah,[1] **Muhammad Sajjad Khan**,[2,3] **Marc St-Hilaire**,[4] **Mohammad Faisal**,[5] **Junsu Kim**,[3] **and Su Min Kim**[3]

[1]*Advanced Technology Research Center, Korea University of Technology and Education, Cheonan, Republic of Korea*
[2]*Department of Electrical Engineering, International Islamic University, Islamabad, Pakistan*
[3]*Department of Electronics Engineering, Korea Polytechnic University, Siheung, Republic of Korea*
[4]*School of Information Technology and Department of Systems and Computer Engineering Carleton University, Ottawa, Canada*
[5]*Department of Computer Science and Information Technology, University of Malakand, Chakdara, Pakistan*

Correspondence should be addressed to Su Min Kim; suminkim@kpu.ac.kr

Received 24 January 2021; Revised 3 March 2021; Accepted 14 March 2021; Published 24 March 2021

Academic Editor: Chien-Ming Chen

The rapid evolution of the Internet of Things (IoT) and the development of cloud computing have endorsed a new computing paradigm called edge computing, which brings the computing resources to the edge of the network. Due to low computing power and small data storage at the edge nodes, the task must be assigned to the computing nodes, where their associated data is available, to reduce overheads caused by data transmissions in the network. The proposed scheme named task priority-based data-prefetching scheduler (TPDS) tries to improve the data locality through available cached and prefetching data for offloading tasks to the edge computing nodes. The proposed TPDS prioritizes the tasks in the queue based on the available cached data in the edge computing nodes. Consequently, it increases the utilization of cached data and reduces the overhead caused by data eviction. The simulation results show that the proposed TPDS can be effective in terms of task scheduling and data locality.

1. Introduction

Edge computing is a paradigm to extend cloud computing services to those at edge nodes in networks. Thus, it brings the computing services near to Internet of things (IoT) devices [1]. Putting resources at the edge of the network enables achieving low latency processing. However, since the enormous number of IoT devices generates a high volume of data, transmitting them to the cloud yields high computational processing. In general, the cloud contains distributed computing resources and processes the data using a group of servers in parallel and distributed way. Sending all the data and tasks to the cloud for processing makes the core network congested and yields a huge load to the cloud servers. To minimize the workload of the core network and the cloud, novel paradigms such as edge computing and fog computing are developed [2–7] to bring computational resources to the edge of the network and offer services near to each IoT device as shown in Figure 1. Due to low computing power and limited data storage, the edge nodes are clustered to perform computation and the huge tasks are distributed to the edge nodes. To distribute the tasks resourcefully and efficiently to the edge nodes based on task-associated data, an efficient task scheduling strategy is required. In other words, a cost-effective task scheduler is needed to assign the tasks closer to the data on a cluster node and bring the resources near to computation nodes while improving the overall system performance.

In cloud computing systems, complicated tasks and data are collected to the cloud for computing processes [8, 9]. All these data and tasks are generated by IoT devices which are connected to the cloud by a middle layer, i.e., IoT edge

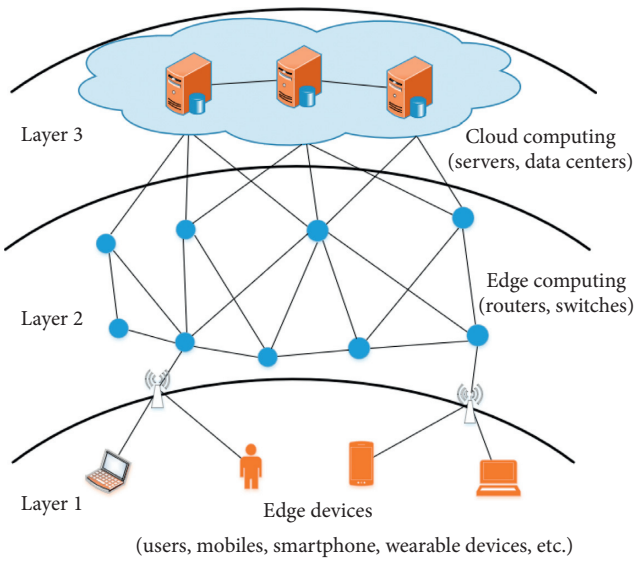

Figure 1: The structure of edge computing network.

nodes. Thousands of IoT devices are connected to the cloud which can yield a heavy load to the core network and the cloud system. This increases the frequency of communication exchanges and causes a long latency to the end-users. Consequently, there are resource limitations in a cloud computing layer to incline many researchers toward the computation at edge devices. Data generated by IoT devices can be processed by middle layer devices such as IoT nodes and base stations. The nodes at the edge level retain low processing power and limited resources and, thus, they cannot handle such heavy and complicated tasks. Therefore, a cost-effective task management strategy is needed to distribute the complicated tasks to the edge nodes in an efficient way.

In a cloud computing cluster, a task manager predicts the amount of data at the computing node and assigns the tasks to appropriate targeted nodes to guarantee data locality [10]. Based on this prediction, each node tries to bring and preload the data from other locations. How much the preloaded data match the task depends on the result of the prediction. The wrong prediction will yield the preloaded data, which is not useful for running the task. It implies the wastes of communication bandwidth and system resources. Yet, the preloaded data can be exploited by fetching the associated task from the queue. So far, several scheduling schemes have been proposed to balance the workload in the network based on the amount of available resource and data [11–20]. All these schemes are prefetching the data but they do not consider tasks' priorities concerning the available cached or stored data. On the contrary, our approach in this paper assigns a priority to a task according to availability if the required data can be obtained from the cached-data queue. Consequently, it can reduce the overhead required for task eviction.

On the other hand, in distributed systems [21, 22], fetching a computation task near to data is cheaper than fetching the data near to the computation task. Bringing the computation task close to the required data is called data locality in cloud computing environments. It is impossible to guarantee 100% data locality but it somehow can be improved with the existing data at the edge level by minimizing unnecessary data transmissions. For quick access, used data are kept in the cache memory for iterative processes. The cache memory contains two different types of data: static data, which is not changeable and can be used in the next round of task execution, and dynamic data, which is changeable and useable in the next round. Due to limited memory capacity, it is impossible to keep all the data needed for the task in the cache memory of the computing node, since data swap-out and swap-in require frequent processes in the cache and storage memories. Loading data from the storage memory to the cache memory is an expensive process in terms of data processing and transferring. If the cache memory becomes full and the system cannot store more data in the cache memory, least recently used (LRU) and first-in-first-out (FIFO) eviction techniques [23] can be applied to swap out unnecessary old data from the cache memory.

In this paper, we extend the idea from our earlier work [11] to utilize the existing preloaded data effectively based on a cost-effective scheduling strategy, named task priority-based data-prefetching scheduler (TPDS), which distributes the tasks to the computing nodes logically. The proposed TPDS tries to match the task in the queue with the cached-data at the computing node. It generates a priority for a task and allocates the task to a proper edge node based on task-associated data in the cache. With this technique, the frequency of data swapping in the cache can be significantly reduced and the data utilization can be improved for available tasks. If there is no task in the queue for the cached-data, the data is swapped out and replaced by the required new data. In this paper, we employ the multi-server queuing theory [24] to evaluate the performance of the proposed scheduling strategy. The proposed TPDS achieves better performance in terms of data locality, task distribution, and reduction of system overheads caused by unnecessary evictions and data exchanges. The main contributions of this paper are summarized as follows:

(i) Dynamic workload scheduling considering queue-wise job priorities is proposed based on data locality of the cache memory in order to maximize the resource efficiency and the data utilization of a cloud cluster.

(ii) In the cloud cluster, our proposed scheme prefetches and evicts the cached-data from the computing node based on task priority. It is able to avoid blind eviction of the cached-data and reduce the system overhead. Hence, it improves the resource efficiency at each node.

(iii) Through assigning a task to the computing node based on the data locality, we can minimize the average completion and waiting time for each task.

(iv) A multi-server queuing model applicable to the proposed TPDS scheme is developed in order to

improve schedulability of the tasks under different constraints and requirements.

The rest of the paper is organized as follows. In Section 2, we review the related previous work concerning scheduling considering prefetching and data locality. We propose a scheduling strategy based on priority-based data-prefetching in Section 3. In Section 4, we evaluate the performance of the proposed strategy, compared to the conventional ones. Finally, this paper is concluded in Section 5.

2. Related Work

Many data locality schemes for task scheduling have been developed to improve the performance of the computing system regarding task execution. The data locality enables avoiding unnecessary data transmissions for the task in cloud computing. In distributed cloud system, tasks are assigned to the nodes in the network based on the prediction of associated data [25].

In [26], a new caching algorithm, called similarity-aware popularity-based caching (SAPoC), is proposed to promote the performance of wireless edge-caching by utilizing the similarity among contents in dynamic scenarios. It is developed for dynamic wireless edge-caching scenarios, where both mobile devices and contents arrive and leave dynamically. In SAPoC, a content's popularity is determined by not only its history of the requests but also its similarity with existing ones to enable a quick-start of newly arrived contents. It aims to devise an efficient edge-caching strategy considering the dynamic nature of wireless edge computing systems.

In [10], data locality aware workflow scheduling (D-LAWS), which focuses on data locality, data transfer time based on network bandwidth, virtual machine (VM) consolidation, and fairness of workflow scheduling at the node level, is proposed. The D-LAWS maximizes resource utilization and parallelism of tasks and analytically formulates data transfer time between VMs. It combines VMs and considers task parallelism by using data flow while planning task executions for a data-intensive scientific workflow. Moreover, it reflects more complex workflow models and the data locality regarding data transfer before task executions. In [27], the authors proposed a novel scheduling scheme for real-time bag-of-tasks jobs that arrive dynamically at a hybrid cloud. It takes into account end-to-end deadlines of the jobs, as well as monetary cost required for use of the complementary public cloud resources. In [28], a novel hierarchical architecture for multiple cloudlets is proposed for mobile edge clouds. In this work, the authors target improving the efficiency of cloud resource utilization by organizing the edge cloud servers into a hierarchical architecture. Instead of serving mobile users directly using a flat collection of edge cloud servers, the basic idea of the proposed scheme is to opportunistically aggregate the mobile loads and send the peak loads exceeding the capacities of edge cloud servers at lower tiers to other servers at higher tiers in the edge cloud hierarchy. They developed analytical models to compare the performance between flat and hierarchical designs of edge computing in terms of resource utilization efficiency. Also, they provided theoretical results that show the advantages of the proposed hierarchical edge cloud architecture.

In [29], Raicu et al. implemented regulating data locality and resource utilization. In [30], the authors proposed a cache-aware task scheduling (CATS) technique that finds suitable resources for executing the data-intensive workload. The proposed model minimizes energy consumptions for both core network and cache accesses. The CATS model brings good tradeoffs between energy minimization and execution time reduction by employing accurate analytical models. Similarly, to enhance the data locality and replication technique, a delay scheduling scheme, called delay scheduling based replication algorithm (DSBRA), is presented in [31]. The DSBRA tries to replicate and de-replicate blocks of the data based on prior information taken from the scheduler. This algorithm focuses on block-level replication but some blocks are stored on the least loaded nodes and some blocks are stored on the heavily loaded nodes. In [32], a locality-based data scheduling algorithm 1 is proposed. It allocates the input data blocks to proper nodes based on their processing capacity in order to enhance the performance of MapReduce in heterogeneous Hadoop clusters.

The prefetching technique is a smart approach for reducing the extra-overhead of data traffic in distributed computing systems. By applying this technique, the delay consumed for task execution can be reduced due to the presence of preloaded data for the task. However, prefetching and predicting data to be preloaded based on the scheduled tasks become a great challenge. In [31, 32], the authors present how to enhance the prefetching techniques and also focus on task scheduling for TaskTracker based on the data. The above-mentioned prefetching strategy maximizes the data locality in distributed computing environments.

Our approach in this paper is based on these previous studies which take into account prefetching to efficiently reuse existing cache data. The main focus in the proposed approach is data eviction and confirmation before task assignment. Our goal is to improve the data locality and to guarantee the resourceful task scheduling in edge computing environments. In the next section, we present the proposed scheduling strategy which enhances the performance of data preloading for tasks and reduces the frequency of cached-data removal blindly. According to our proposed approach, the task scheduler tries to select the most appropriate node in the edge computing cluster from the perspective of the data locality and to assign the task to the selected node. It is able to increase the cached-data utilization and enhance the swapping process for minimizing the overall system overhead.

3. Proposed Task Priority-Based Data-Prefetching Scheduler (TPDS)

In this section, the proposed TPDS is presented for edge computing clusters. The TPDS tries to avoid unnecessary eviction of data in order to improve the operation process of task scheduling and data caching. Since the costs of data transfer and eviction result in a great impact on system

performance, the proposed TPDS attempts to reduce the costs for data transfer and eviction, while it tries to improve the task execution procedure.

3.1. Design Goals. The design goals of the proposed strategy are (i) prioritization of tasks based on the existing data in the cache memory of the computing node, (ii) improvement of awareness between the computing nodes and a task manager regarding data and task to increase hit ratio of the cached-data, and (iii) speeding up the execution of tasks by reducing the waiting time of jobs and increasing the utilization of the cached-data. Let us consider a set of tasks $T = \{t_1, t_2, t_3, \ldots, t_n\}$ with the associated data set $D = \{d_1, d_2, d_3, \ldots, d_n\}$ and edge computing nodes $E = \{e_1, e_2, e_3, \ldots, e_n\}$, which contain different data blocks d_n in the cache memory, C, or storage, S. Based on the traditional data locality scheme, the task $t_n \in T$ will be assigned to the computing node $e_n \in E$ which contains its required data, d_n. Then, task allocation to the node can be expressed as

$$t_n \longrightarrow e_n \begin{cases} S_{e_n \in E} \exists d_n, \\ or \\ S_{e_n \in E} \leftarrow R_{L_n} \exists d_n, \end{cases} \tag{1}$$

where R_{L_n} denotes any remote location, which contains data d_n near to node e_n.

We assume that five tasks arrive in the system as shown in Figure 2. The details of task allocation to the computing node, $e_n \in E$, are given in Table 1. The task t_1 is assigned to the computing node e_1 since the cached-data of the node, $C_{e_1 \in E}$, have the data block d_8 that is needed for the processing of t_1. Similarly, the task t_2 needs d_2 which is unavailable in the cache of the n-th node, $C_{e_n \in E}$, but available in the storage $S_{e_2 \in E}$ of the node e_2. By the LRU cache replacement policy, d_0, which is the old data block, is swapped with d_2. Similarly, the task t_3 needs the data block d_3, which considers an old data block is replaced with d_1 as shown in Figure 2.

In Figure 2, it is noted that there are two data blocks d_0 and d_3 which are replaced with d_2 and d_1 by the LRU policy for the tasks t_2 and t_3, respectively, due to the limited capacity of the cache memory. After finishing, the tasks t_2 and t_3, and the data blocks d_0 and d_3 will shift again to the cache $C_{e_n \in E}$ for the tasks t_4 and t_5, which require them. Therefore, the proposed scheduling strategy avoids such unnecessary eviction and swapping of data by prioritizing the tasks based on the available cached-data in the computing node $C_{e_n} \exists d_n$ as shown in Table 2 and Figure 3. Equations (2) and (3) express the computing node and task allocation based on the availability of cached-data.

$$e_n \in E = \forall e_n \left(C_{e_n \in E}, S_{e_n \in E} \right), \tag{2}$$

$$\forall t_n \longrightarrow \forall e_n \begin{cases} C_{e_n \in E} \exists d_n, \\ S_{e_n \in E} \exists d_n, \\ S_{e_n \in E} \leftarrow R_{L_n} \exists d_n. \end{cases} \tag{3}$$

4. Performance Evaluation Model

In this section, a theoretical model of the proposed TPDS is formulated and derived. We employ an $M/M/c$ queuing model to evaluate the performance of the proposed TPDS. Suppose that there are n number of tasks denoted by $T = \{t_1, t_2, t_3, \ldots, t_n\}$ with a set of data blocks denoted by $D = \{d_1, d_2, d_3, \ldots, d_n\}$ and a set of computing nodes denoted by $E = \{e_1, e_2, e_3, \ldots, e_m\}$. Here, e denotes the computing nodes, m represents the total number of computing nodes, D represents the set of data blocks, and d_n denotes the specific data block required for a task. If all tasks arrive in the system, the total number of data blocks contained in the cache for all computing nodes can be expressed as

$$T = \sum_{i=1}^{n} t_i, \tag{4}$$

$$D = \sum_{i=1}^{m} \sum_{j=1}^{n} d_j. \tag{5}$$

According to the proposed TPDS, before eviction of the data $d_n \in D$ from the cache memory $C_{e_m \in E}$, the computing node sends a request to the task manager in order to know if there is any task $t_n \in T$ in the queue for this eviction of the data $d_n \in D$. If there is a task in the queue of the task manager, then it gives a priority to the task and assigns that task to the node $e_m \in E$. Otherwise, the data d_n is evicted and swapped in the cache memory. To estimate and optimize the probabilistic performance of edge computing nodes, the notations are defined in Table 3.

In this model, we consider two types of tasks: high priority task and low priority task, based on the cached-data as shown in Figure 3. The high priority tasks are the tasks whose required data are already available in the cache memory and low priority tasks are the tasks whose required data are not available in the cache memory of the edge node $e_n \in E$ as follows:

$$t_n = \begin{cases} t_{n < C_{e_n \in E} \exists d_n} > \text{high priority}, \\ or \\ t_{n < C_{e_n \in E} \nexists d_n} > \text{low priority}. \end{cases} \tag{6}$$

We consider all tasks arriving at the edge computing nodes with the rate of $\lambda \in T$. We assume that the arrival of a

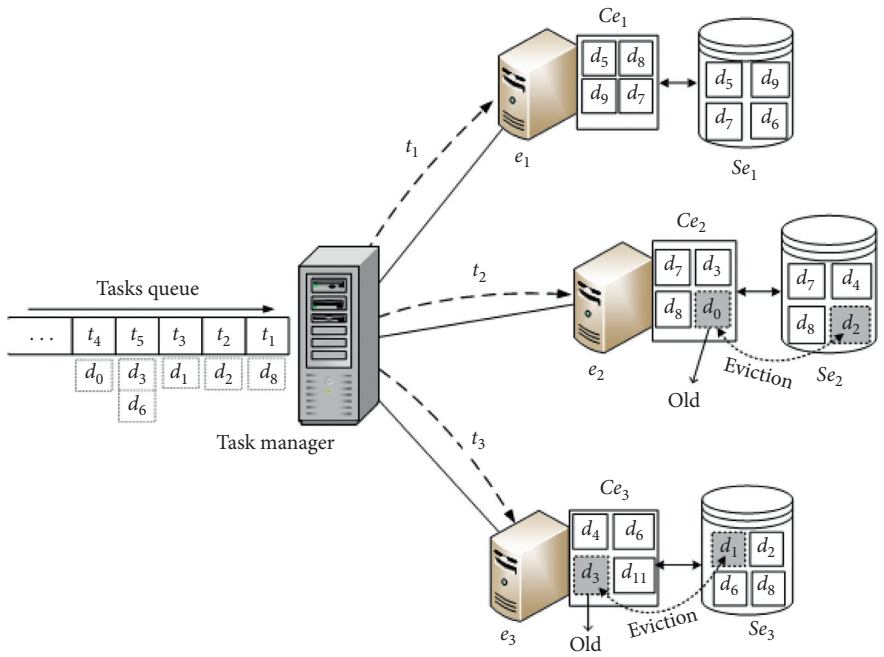

Figure 2: Data prefetching and eviction process without task priority.

```
(1) Initialize the values
(2)   QT: Queue of tasks in TaskManager.
(3)   QD: Record of cached-data in edge node.
(4)   QE: List of nodes
(5)   Qtn.dn: List of data needed for task execution.
(6) Procedure:
(7)   Check status of nodes
(8)     e_n ← Idle
(9)     e_n ← busy
(10)  While (QT is not empty) do
(11)    if (e_n is idle) then
(12)      for all tasks in queue do
(13)        if tn.dn ∈ C_{e_n∈E} then
(14)            e_n ∈ E ← tn <h>
(15)          else
(16)              if (C_{e_n∈E} need eviction) then
(17)                  evcit ← C_{e_n∈E}.old_data
(18)                  C_{e_n∈E} ← S_{e_n∈E} d_n
(19)                  e_n ∈ E ← t_n
(20)              end if
(21)          end if
(22)      end for
(23)        busy ← e_n
(24) end if
(25) end while
```

Algorithm 1: Task priority-based data-prefetching.

task follows a Poisson process and each arrival is transferred to different nodes in the cluster of edge computing nodes. Let $\rho = \lambda/\mu$ be the traffic strength regarding the tasks with different priorities based on the available cached-data, where λ and μ are the arrival rate and the service rate, respectively.

The parameters for task requests in the queuing model are N_s, W_Q, and T_s. Among these three parameters, W_Q affected by the number of tasks being served plays the primary role in the performance. As shown in Figure 4, the scheduling policy is based on $M/M/(e_n \in E)$. According to $M/(e_n \in E)$

Table 1: An example of assigning tasks without considerations of priority and data locality.

Arrival of tasks	Required data	Computing nodes
t_1	d_8	$e_1 \exists d_8$
t_2	d_2	$e_2 \exists d_2$
t_3	d_1	$e_3 \exists d_1$
t_4	d_0	$e_2 \exists d_0$
t_5	$d_3 + d_6$	$e_2 \exists d_{3+6}$

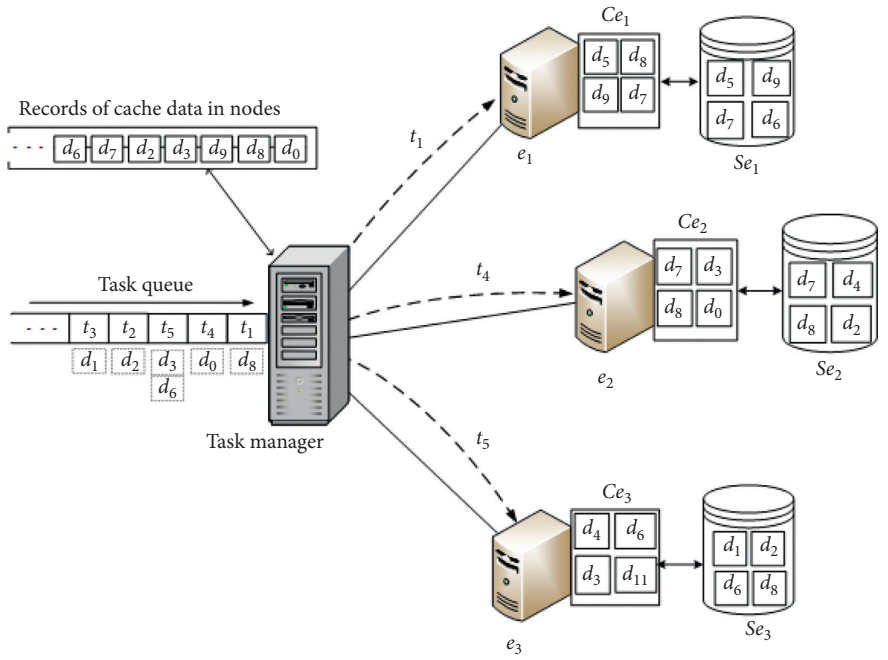

Figure 3: Data prefetching and eviction process based on task priority.

Table 2: An example of assigning tasks based on priority and data locality.

Arrival of tasks	Prioritized tasks	Required data	Computing nodes
t_1	t_1	d_8	$e_1 \exists d_8$
t_2	t_4	d_0	$e_2 \exists d_0$
t_3	t_5	$d_3 + d_6$	$e_2 \exists d_{3+6}$
t_4	t_2	d_2	$e_2 \exists d_2$
t_5	t_3	d_1	$e_3 \exists d_1$

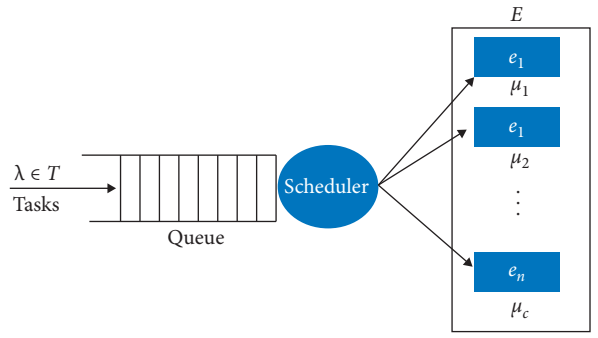

Figure 4: Edge computing queue model $M/M/(e_n \in E)$.

queueing model, remaining time, waiting time, and service time of the tasks in edge computing are mathematically evaluated.

As the requests to edge nodes come from the end devices like smartphones, tablets, and wearable devices, the pool of the tasks and the size of the queue are considered to be limitless in the task manager at the cluster of edge nodes. The state transition diagram of $M/M/(e_n \in E)$, which can be denoted through balance equations, is shown in Figure 5. When the number of tasks $t_n \in T$ is less than the number of computing nodes $e_n \exists d_n$, only n out of the nodes e_n are busy and the mean service rate is equal to n?. From (4), we can obtain

$$P_n = P_0\left[\frac{(e_n \exists d_n \rho)^n}{(n)!}\right] \quad (1 \le n \le e_n). \tag{7}$$

If the number of tasks is greater than or equal to $e_n \exists d_n$, i.e., $n \ge e_n \exists d_n$, all the nodes are busy and the effective service rate is equal to $\mu(e_n \exists d_n)$. Thus,

$$P_n = P_0\left[\frac{(e_n \exists d_n \rho)^n}{(e_n \exists d_n)^{n-P}(e_n \exists d_n)!}\right] \quad \text{for } n > (e_n \exists d_n). \tag{8}$$

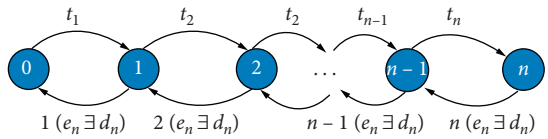

Figure 5: State transition diagram of $M/M/(e_n \in E)$.

Here, $\rho = \lambda/\mu(e_n \exists d_n)$ where ρ must be less than 1 for system stability. Note that the expected number of busy nodes is equal to $\rho(e_n \exists d_n) = \lambda/\mu$. To obtain P_0, both sides of (7) and (8) are summed up. Since $\sum_{n=0}^{\infty} P_n = 1$, P_0 is derived as

$$P_0 = \left[\sum_{n=1}^{e_n-1} \frac{(e_n \exists d_n \rho)^n}{n!} + \frac{\rho^{e_n}}{(e_n \exists d_n)!(1-\rho)}\right]^{-1}. \quad (9)$$

The proposed TPDS is an efficient scheduling strategy that minimizes the costs of data transfer and execution latency. For evaluation of the system performance, it is necessary to calculate the total number of tasks in the queue, the total waiting time, the service time of the jobs, and the total number of tasks in the system. If the number of incoming tasks is less than the number of nodes in the cluster as represented in (7), the system is under the stable condition. Thus, it is expected that all tasks can be completed on time with no extra waiting in the queue. Otherwise, as in (8), it is highly probable that some tasks wait for long time and never get a service. The proposed TPDS tries to optimally minimize unnecessary eviction and improve the data locality for the tasks. As discussed earlier, when $n > e_n \exists d_n$, some tasks must wait in the queue. Thus, the estimated number of tasks in the queue is given by

$$N_Q = P_0 \frac{(\rho)^{e_n \exists d_n}}{(e_n \exists d_n - 1)!(\mu e_n \exists d_n - \lambda)^2}. \quad (10)$$

To evaluate the system performance by applying Little's law, it is necessary to obtain the total waiting time of tasks before service, the total number of tasks in the queue, and the total time spent by a single task in the cluster of edge computing nodes.

$$W_Q = P_0 \frac{\mu(\rho)^{e_n \exists d_n}}{(e_n \exists d_n - 1)!(\mu e_n \exists d_n - \lambda)^2}, \quad (11)$$

$$T_s = P_0 \left[\frac{\mu(\rho)^{\mu e_n \exists d_n}}{(e_n \exists d_n - 1)!(\mu e_n \exists d_n - \lambda)^2}\right] + \frac{1}{\mu}, \quad (12)$$

$$N_s = P_0 \left[\frac{\lambda\mu(\rho)^{e_n \exists d_n}}{(\mu e_n \exists d_n - 1)!(\mu e_n \exists d_n - \lambda)^2}\right] + \frac{\lambda}{\mu}. \quad (13)$$

The probability that all nodes are busy in edge computing clusters can be derived from (14) and (15).

$$P_B = P_0 \sum_{n=e_n}^{\infty} \left(\rho^n \frac{(e_n \exists d_n)^{e_n}}{(e_n \exists d_n)!}\right), \quad (14)$$

$$P_B = P_0 \frac{(e_n \exists d_n)^{e_n}}{(e_n \exists d_n - 1)!(\mu e_n \exists d_n - \lambda)}. \quad (15)$$

5. Performance Evaluation

In this section, the proposed TPDS is evaluated through computer simulations. The job completion time and node utilization under data locality in the cache memory are estimated by Cloudsim [33]. Cloudsim includes a broker (task manager node) and client nodes (number of machines) entities. The results of the proposed TPDS are compared with the existing scheduling and eviction schemes: FIFO, LRU, and HPSO [23, 34]. The efficiency of the proposed TPDS strategy is evaluated in terms of hit ratio of cached-data, task execution time, task waiting time, and data locality. The parameter details for the Cloudsim simulator are given in Table 4.

Figure 6 shows the used ratio of data for the proposed TPDS, compared with the three conventional schemes. The proposed TPDS maximizes utilization of the cached-data by using it for the incoming task in the queue. The proposed TPDS is not blindly swapping out the old data without knowing the incoming task in the queue. Thus, it is noted that the hit ratio is higher in the proposed TPDS than the conventional FIFO, LRU, and HPSO schemes. Particularly, when the number of data blocks increases, the time consumed for completing the task for all the schemes also increases due to swapping out data blocks from the cache memory without checking the queue of the task manager. This causes lower hit ratios in the cached-data as the number of data blocks increases.

Figure 7 shows the execution times of tasks of the proposed TPDS and three conventional schemes. As shown in Figure 8, the execution time of the task for the proposed TPDS is always smaller than those of the conventional FIFO, LRU, and HPSO schemes. It is because the more swapping out of data gives the longer waiting time to the task to update the associated data for the incoming task. Pre-existing data for tasks will execute the task quickly and there is no need to wait to bring the related data.

Similarly, Figure 8 shows the average waiting time of tasks. From the figure, it is observed that the waiting time of the proposed TPDS is smaller than those of the conventional FIFO, LRU, and HPSO schemes. Compared to the conventional schemes, the proposed TPDS consistently allows shorter average waiting time in the whole range of the number of tasks. The number of tasks is varying from 200 to 2200 and the same distribution of job sizes is maintained throughout the simulation test. The proposed TPDS significantly outperforms the other conventional schemes in terms of average execution time as the number of tasks increases.

Table 3: The variables used in the performance model.

Variable	Description
N_Q	The number of tasks waiting in the queue for service
N_s	Total number of tasks in the edge computing system
T_s	The total time spent by the task in the edge computing system
W_Q	The total time of the task waiting in the queue for service
P_n	The probability that the system has "n" number of tasks
P_b	The probability that all nodes are busy in the edge computing system
$e_n \exists d_n$	The computing node that contains the related data, d_n, for the task

Table 4: Cloudsim simulator parameters.

Entity	Parameters	Values
Cloudlets/task	Length of the task (expected number of instructions)	50–2000
	Number of tasks for six times running	200 to 2200
	The priority of tasks based on the cached-data blocks	High, medium, and low
Virtual machine	Number of VMs	50 in ache data center
	VM memory	1 GB to 2 GB
	Bandwidth	500–1000
	Number of CPUs	1 to 3
Data center	Number of data centers	5
	Number of hosts	2 to 3

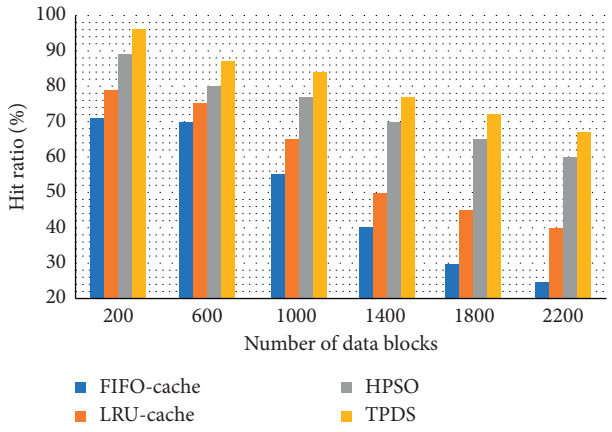

Figure 6: The comparison of hit ratio.

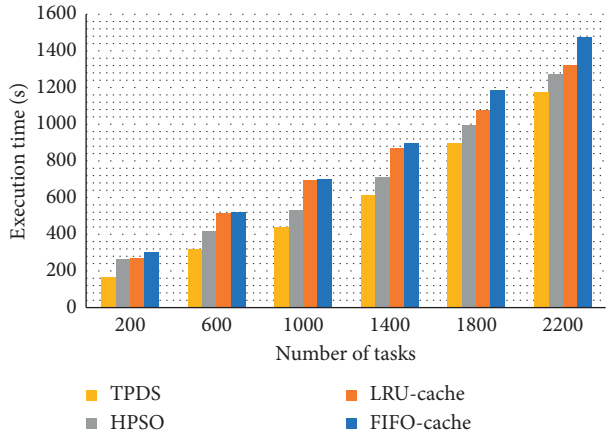

Figure 7: The comparison of task execution time.

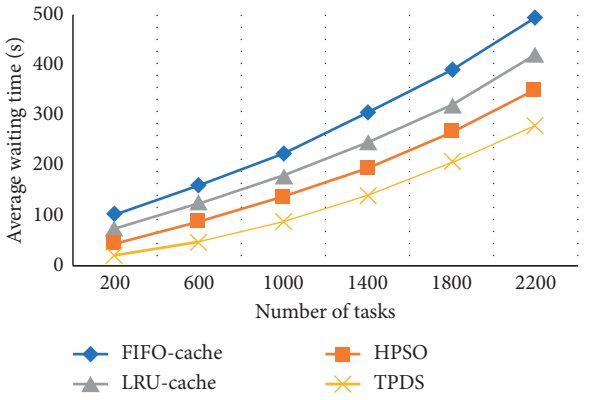

Figure 8: The comparison of average waiting time.

Another feature of the proposed TPDS is the priority scheduling of tasks as shown in Figure 9. It is noted that, with help of this scheduling strategy, the jobs achieve nearly the best data locality, which is helpful to improve the performance of distributed systems. The proposed TPDS takes advantage of cached-data locality to accelerate the computation of the task and minimize the CPU usage and data transfer load in terms of swapping out and swapping in data from the cache memory. It significantly improves the performance of the computing nodes and the execution of tasks. It is shown that the proposed TPDS also always outperforms the conventional schemes in terms of data locality.

In Figure 10, the average execution time of the proposed TPDS is compared with the conventional schemes. We use six different types of workloads as different numbers of data blocks (200 to 2200). Compared to the conventional schemes, the proposed TPDS reduces the average execution time by 8.5% to 10.2% for six different workloads,

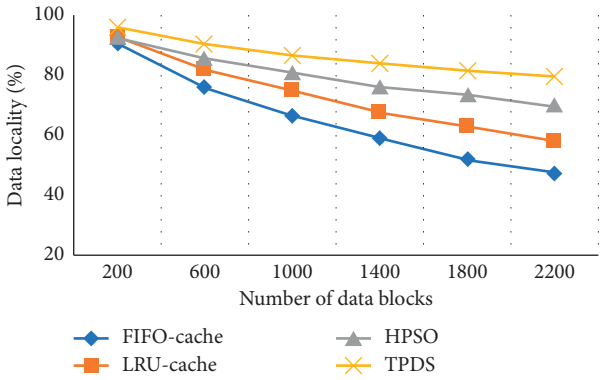

Figure 9: The comparison of data locality.

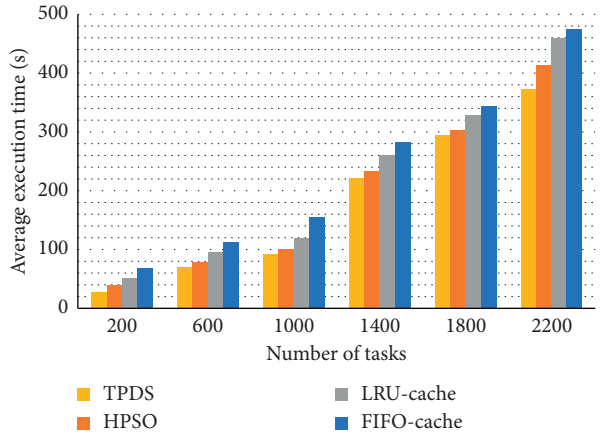

Figure 10: The comparison of average execution time.

respectively. This demonstrates that the proposed TPDS performs data locality more efficiently than the existing schemes due to the availability of data blocks in the cache memory.

6. Conclusion

As the number of IoT devices and the scale of cloud computing grow in popularity, many edge computing and distributed systems have emerged in recent years. In general edge computing architecture, computing power, bandwidth, and data at the edge are scarce resources. To improve system performance, a task scheduling strategy must be efficient. In this paper, we proposed a cache data locality scheduler for edge-computing cluster environments. The proposed strategy schedules tasks by taking a broad view and adjusts data for tasks dynamically according to data in cache memory. Especially in an edge computing cluster environment, where the number of resources is limited, our proposed approach tries its best to enhance task execution under limited resources and reduce the extra flow of data in the cluster network. When the computing cluster is overloaded, the proposed strategy takes the advantage of data in the cache and brings the task first which finds the needed data in the cache of the node. The simulation results show that the proposed strategy exhibits some improvements which can also work in a busy network and cluster. As future work, we plan to improve the proposed task scheduling strategy based on available resources. We will consider the aspects that may affect the performance including data distributions and replication in a heterogeneous system. Edge computing and distributed technologies are growing up due to massive data volume generated by a large number of IoT devices. Accordingly, it is essential to keep update and development on scheduling strategies and efficient algorithms for tasks to manage resources in edge computing environments.

Data Availability

The data used to support the findings of this study are included within this article.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-2018-0-01426) supervised by the IITP (Institute for Information and Communication Technology Planning & Evaluation) and in part by the National Research Foundation (NRF) funded by the Korea Government (MSIT) (no. 2019R1F1A1059125).

References

[1] B. Liu, H. Xu, and X. Zhou, "Resource allocation in wireless-powered mobile edge computing systems for internet of things applications," *Electronics*, vol. 8, no. 2, p. 206, 2019.

[2] Z. Xu, W. Liu, J. Huang, C. Yang, J. Lu, and H. Tan, "Artificial intelligence for securing IoT services in edge computing: a survey," *Security and Communication Networks*, vol. 2020, 2020.

[3] P. Garcia Lopez, A. Montresor, D. Epema et al., "Edge-centric computing," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[4] T. H. Luan, L. Gao, Z. Li, Y. Xiang, and L. Sun, "Fog computing: focusing on mobile users at the edge," 2015, https://arxiv.org/abs/1502.01815.

[5] I. Ullah and H. Y. Youn, "Task classification and scheduling based on K-means clustering for edge computing," *Wireless Personal Communications*, vol. 113, no. 4, pp. 2611–2624, 2020.

[6] J. Bellendorf and Z. Á. Mann, "Classification of optimization problems in fog computing," *Future Generation Computer Systems*, vol. 107, pp. 158–176, 2020.

[7] S. A. Chaudhry, K. Yahya, F. Al-Turjman, and M.-H. Yang, "A secure and reliable device access control scheme for IoT based sensor cloud systems," *IEEE Access*, vol. 8, pp. 139244–139254, 2020.

[8] L. M. Dang, M. J. Piran, D. Han, K. Min, and H. Moon, "A survey on internet of things and cloud computing for healthcare," *Electronics*, vol. 8, no. 7, p. 768, 2019.

[9] D. C. Marinescu, *Cloud Computing: Theory and Practice*, Morgan Kaufmann, Burlington, MA, USA, 2017.

[10] J. Choi, T. Adufu, and Y. Kim, "Data-locality aware scientific workflow scheduling methods in HPC cloud environments," *International Journal of Parallel Programming*, vol. 45, no. 5, pp. 1128–1141, 2017.

[11] I. Ullah, M. S. Khan, M. Amir, J. Kim, and S. M. Kim, "LSTPD: least slack time-based preemptive deadline constraint scheduler for Hadoop clusters," *IEEE Access*, vol. 8, pp. 111751–111762, 2020.

[12] C.-H. Chen, T.-Y. Hsia, Y. Huang, and S.-Y. Kuo, "Scheduling-aware data prefetching for data processing services in cloud," in *Proceedings of the 2017 IEEE 31st International Conference on Advanced Information Networking and Applications*, pp. 835–842, Taipei, Taiwan, March 2017.

[13] M. Sun, H. Zhuang, C. Li, K. Lu, and X. Zhou, "Scheduling algorithm based on prefetching in MapReduce clusters," *Applied Soft Computing*, vol. 38, pp. 1109–1118, 2016.

[14] C. Li, J. Zhang, Y. Chen, and Y. Luo, "Data prefetching and file synchronizing for performance optimization in Hadoop-based hybrid cloud," *Journal of Systems and Software*, vol. 151, pp. 133–149, 2019.

[15] C.-H. Chen, T.-Y. Hsia, Y. Huang, and S.-Y. Kuo, "Data prefetching and eviction mechanisms of in-memory storage systems based on scheduling for big data processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1738–1752, 2019.

[16] H. Cui, X. Liu, T. Yu, H. Zhang, Y. Fang, and Z. Xia, "Cloud service scheduling algorithm research and optimization," *Security and Communication Networks*, vol. 2017, 2017.

[17] A. Samanta, Z. Chang, and Z. Han, "Latency-oblivious distributed task scheduling for mobile edge computing," in *Proceedings of the 2018 IEEE Global Communications Conference*, pp. 1–7, Abu Dhabi, United Arab Emirates, December 2018.

[18] Y. Deng, Z. Chen, X. Yao, S. Hassan, and J. Wu, "Task scheduling for smart city applications based on multi-server mobile edge computing," *IEEE Access*, vol. 7, pp. 14410–14421, 2019.

[19] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4854–4866, 2018.

[20] S. A. Chaudhry, I. L. Kim, S. Rho, M. S. Farash, and T. Shon, "An improved anonymous authentication scheme for distributed mobile cloud computing services," *Cluster Computing*, vol. 22, no. 1, pp. 1595–1609, 2019.

[21] M. Wang and Q. Zhang, "Optimized data storage algorithm of IoT based on cloud computing in distributed system," *Computer Communications*, vol. 157, pp. 124–131, 2020.

[22] L. M. Haji, S. Zeebaree, O. M. Ahmed, A. B. Sallow, K. Jacksi, and R. R. Zeabri, "Dynamic resource allocation for distributed systems and cloud computing," *TEST Engineering & Management*, vol. 83, pp. 22417–22426, 2020.

[23] K. Lu, D. Dai, X. Zhou, M. Sun, C. Li, and H. Zhuang, "Unbinds data and tasks to improving the hadoop performance," in *Proceedings of the 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 1–6, Melbourne, Australia, August 2014.

[24] L. Guo, T. Yan, S. Zhao, and C. Jiang, "Dynamic performance optimization for cloud computing using M/M/m queueing system," *Journal of Applied Mathematics*, vol. 2014, Article ID 756592, 18 pages, 2014.

[25] G. Chen, S. Wu, R. Gu et al., "Data prefetching for scientific workflow based on Hadoop," *Computer and Information Science 2012*, vol. 429, pp. 81–92, 2012.

[26] X. Wei, J. Liu, Y. Wang, C. Tang, and Y. Hu, "Wireless edge caching based on content similarity in dynamic environments," *Journal of Systems Architecture*, vol. 115, Article ID 102000, 2021.

[27] G. L. Stavrinides and H. D. Karatza, "Dynamic scheduling of bags-of-tasks with sensitive input data and end-to-end deadlines in a hybrid cloud," *Multimedia Tools and Applications*, pp. 1–23, 2020.

[28] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proceedings of the IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, April 2016.

[29] I. Raicu, I. T. Foster, Y. Zhao et al., "The quest for scalable support of data-intensive workloads in distributed systems," in *Proceedings of the 18th ACM international symposium on High performance distributed computing-HPDC '09*, pp. 207–216, Munich, Germany, June 2009.

[30] S. N. Prasad, S. Kulkarni, and P. Venkatareddy, "Cache aware task scheduling algorithm for heterogeneous cloud computing environment," in *Proceedings of the Fifth International Conference on Research in Computational Intelligence and Communication Networks*, pp. 154–158, Kolkata, India, March 2020.

[31] S. Suresh, N. Gopalan, and N. P. Gopalan, "Delay scheduling based replication scheme for Hadoop distributed file system," *International Journal of Information Technology and Computer Science*, vol. 7, no. 4, p. 73, 2015.

[32] N. S. Naik, A. Negi, T. B. B.R., and R. Anitha, "A data locality based scheduler to enhance MapReduce performance in heterogeneous environments," *Future Generation Computer Systems*, vol. 90, pp. 423–434, 2019.

[33] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[34] M. Sun, H. Zhuang, X. Zhou, K. Lu, and C. Li, "HPSO: prefetching based scheduling to improve data locality for mapreduce clusters," *Algorithms and Architectures for Parallel Processing,Lecture Notes in Computer Science*, vol. 8631, pp. 82–95, 2014.

[35] B. Jiang, J. Wu, X. Shi, and R. Huang, "Hadoop scheduling base on data locality," 2015, https://arxiv.org/abs/1506.00425.