energies

Article

Data-Intensive Task Scheduling for Heterogeneous Big Data Analytics in IoT System

Xin Li [1], Liangyuan Wang [1], Jemal H. Abawajy [2], Xiaolin Qin [1], Giovanni Pau [3] and Ilsun You [4,*]

1 College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210023, China; lics@nuaa.edu.cn (X.L.); myteng@nuaa.edu.cn (L.W.); qinxcs@nuaa.edu.cn (X.Q.)
2 School of Information Technology, Deakin University, Locked Bag 20000, Geelong, VIC 3220, Australia; jemal.abawajy@deakin.edu.au
3 The Faculty of Engineering and Architecture, Kore University of Enna, 94100 Enna, Italy; giovanni.pau@unikore.it
4 Department of Information Security Engineering, Soonchunhyang University, Asan-si 31538, Korea
* Correspondence: isyou@sch.ac.kr

Received: 13 July 2020; Accepted: 17 August 2020; Published: 1 September 2020

Abstract: Efficient big data analysis is critical to support applications or services in Internet of Things (IoT) system, especially for the time-intensive services. Hence, the data center may host heterogeneous big data analysis tasks for multiple IoT systems. It is a challenging problem since the data centers usually need to schedule a large number of periodic or online tasks in a short time. In this paper, we investigate the heterogeneous task scheduling problem to reduce the global task execution time, which is also an efficient method to reduce energy consumption for data centers. We establish the task execution for heterogeneous tasks respectively based on the data locality feature, which also indicate the relationship among the tasks, data blocks and servers. We propose a heterogeneous task scheduling algorithm with data migration. The core idea of the algorithm is to maximize the efficiency by comparing the cost between remote task execution and data migration, which could improve the data locality and reduce task execution time. We conduct extensive simulations and the experimental results show that our algorithm has better performance than the traditional methods, and data migration actually works to reduce th overall task execution time. The algorithm also shows acceptable fairness for the heterogeneous tasks.

Keywords: big data analysis; heterogeneous data-intensive task; IoT system; service response delay; task scheduling

1. Introduction

Big data analysis plays an important role in many IoT application scenarios, it supports various services for users. The cloud data center is the promising way to deal with the big data analysis tasks and support data-intensive [1] services. However, it is becoming hard due to the big data analysis tasks are becoming heterogeneous, it may be periodic data analysis tasks for enterprises to analyze user behaviors, or online tasks from edge environment [2] to support various services [3]. For these tasks, it has different expectation on task execution time. While scheduling heterogeneous tasks, the system not only needs to reduce the execution time for periodic tasks, but also response online tasks as quickly as possible, usually within a few minutes, even a few seconds [4]. Hence, it is an important issue to schedule the heterogeneous tasks in data center to support various services, which is also important for energy reduction since longer task execution also consumes more energy.

Researchers have proposed many approaches to meet the demand for periodic tasks or online tasks [5]. Since the scenarios of applying big data are getting more and more complicated, only scheduling periodic tasks or online tasks cannot satisfy the needs of users. However, the computing resources in data centers are not unlimited, periodic tasks and online tasks have to compete for resources. While scheduling the heterogeneous tasks consist of periodic tasks and online tasks simultaneously, it is difficult to reduce the execution time of periodic tasks and respond online tasks timely. In this paper, we propose an algorithm to solve this problem of scheduling heterogeneous tasks and compare it with some classical task scheduling methods. To reduce the task execution time, we prefer the task is executed in locality mode, which means the task can access the required data from the server which hosts the task. It is easy-understanding that remote data access is time consuming for network delay. Otherwise, the task will be executed in remote mode, which indicates the task will access the data remotely. To reduce task execution time, it is preferred to schedule a task from the queue to occupy the idle resource with locality mode. However, it is impossible to achieve locality mode for all tasks due to the limited resource. The delay scheduling [6] is proposed to achieve more data locality by delaying the task execution until the desired server has idle resource. It is an efficient method to improve the data locality, and this also motivate us move the data from one server to another actively, but not passively like the delay scheduling.

In this paper, we investigate the task scheduling problem, and propose the task scheduling algorithm for heterogeneous tasks with data migration. We take the data block as an important part since it affects the data locality directly. It is acceptable that we cannot schedule all tasks to be executed in locality mode since the storage space is limited in data centers. The competition for computing resources between periodic tasks and online tasks is also a problem. However, some traditional task scheduling methods are no longer applicable to heterogeneous tasks like FIFO. To satisfy the demands of time reduction, in our algorithm, we first established a time model for time calculation of tasks executed in different modes based on some of our previous work [7]. In order to execute tasks in locality mode as much as possible, we migrate the required data block to the server when scheduling a task. We also compare the heterogeneous task scheduling algorithm with other classical algorithms and conduct some simulation experiments. The results show that our algorithm can guarantee better data locality and acceptable fairness for tasks so that improve the efficiency of whole system.

In summary, we make the following contributions in this paper.

- We formulate the scheduling problem for heterogeneous tasks in data centers, and propose a data-migration based algorithm to achieve better data locality for task scheduling. To the best of our knowledge, this is the first attempt to introduce the active manner to schedule both periodic tasks and online tasks simultaneously.
- The proposed algorithm take both delay scheduling and data migration into account to improve the data locality. We establish the task execution model to predict the task execution time, and make measurable comparison for various execution cost to achieve better decision.
- We conduct extensive simulation experiments to evaluate the efficiency of the algorithm. The results show that the algorithm has better performance on task execution time reduction. In addition, the fairness of the tasks is also acceptable.

The rest of this paper is organized as follows. Section 2 describes the related work on task scheduling and data deployment. Section 3 introduces how to establish the model of server selection and how to calculate different time consumptions. Then, we introduce the heterogeneous task scheduling strategy in Section 4. In Section 5, we conduct some experiments and evaluate the efficiency of the algorithm. Finally, we conclude this paper in Section 6.

2. Related Work

The research community have conducted the task scheduling from various aspects, like makespan reduction, resource sharing, energy saving, fault tolerance and so on. In addition to the scheduling in cloud data centers, it is also a hot topic in Internet of Things (IoT) system [8–10], and Mobile Edge Computing (MEC) [11,12]. In this paper, we focus on the big data analysis task scheduling issue in cloud data centers. For the data-intensive tasks, except the resources, data locality is the core issue to improve the task scheduling efficiency, it greatly affects the execution time of tasks [13].

MapReduce [14] is the typical data processing paradigm for big data analysis. For the Hadoop [15], the opensource implementation for MapReduce, the default task scheduling strategy is First In First Out (FIFO), which is simple and has low overhead. However, it is unacceptable for the heterogeneous tasks with different Quality of Service (QoS) requirements, and the performance cannot be guaranteed since the data distribution could be not suitable for task execution due to frequent data transmission. Hence, Fair Scheduler is proposed based on multi-user environment [16]. This scheduling method sets a pool of resources for each user in cluster which can ensure that the users can occupy the same amount of resource with high probability. The Fair Scheduler support some kind fairness among different tasks, and also brings resource waste or resource shortage since the tasks have heterogeneous resource requirements, which limits the performance improvement for Fair Scheduler. To improve the resource utilization, Capacity Scheduler [16] is proposed by providing resource-aware task scheduling. The core idea is to adjust the tasks in the queue by sensing the resource usage case.

Most of the former task scheduler, like ShuffleWatcher [17], BAR [18], LATE [19], and SAMR [20], mainly focus on the resources, especially for the computing resources, since the computation with CPU is the main work for the task. However, there will be frequent I/O operation for the big data analysis tasks, which make the data locality more important since the local data I/O could reduce the task execution time efficiently than remote data access. Delay scheduling [6] is the representative scheduler to schedule the online tasks dynamically by achieving better data locality. The core idea is to let the task waits for a few moment if the server with the desired data has no sufficient computing resource until some task release resources or reach the threshold time. The method works since it can achieve data locality with more opportunities. We can also regard it makes a tradeoff between the waiting time and the data transmission time.

The delay scheduling could be a passive method to achieve data locality for online task scheduling. Since the data placement is important for data locality, there are also some work focus on the data placement issue to provide more data locality opportunity. CoHadoop [21], Scarlett [22], and PACMan [23] are presented to place the data reasonably for achieving better data locality. The joint task and data placement is another approach to achieve data locality for offline tasks [24], which analyzes the relevancy between task and data to decide how to arrange the data placement and task assignment comprehensively. In addition, with the different idea, the migration approach [7] is proposed to achieve data locality actively. The main idea is to compare the cost between waiting cost and migration cost. The data will be moved from some server to the desired server if the data migration can achieve data locality and with less cost.

As more and more date is generated in the mobile edge devices, the task scheduling issue will be discussed in edge computing environment. The cache-aware task scheduling method in edge computing is proposed to achieve better data locality [25], which obeys the similar idea is this paper. This trend also prompt the task scheduling issue in System on a Chip (SoC), an energy-aware task scheduling is proposed for heterogeneous real-time MPSoCs in IoT [26].

For the most of the approaches discussed above, they work for scheduling only periodic tasks or only online tasks, and work in passive manner. In this paper, we propose the scheduling strategy to schedule heterogeneous tasks consist of periodic and online tasks with active manner, which is a new attempt for task scheduling. The comparison of the features is shown in Table 1.

For the most scheduling strategies discussed above are applied to schedule only periodic tasks or only online tasks. They cannot satisfy the demand for scheduling both two kinds of tasks simultaneously. Hence, we propose our scheduling strategy to schedule heterogeneous tasks consist of periodic and online tasks and take the input data into consideration.

Table 1. The feature comparison for representative approaches.

Approach	Periodic Task	Online Task	Passive Manner	Active Manner
FIFO	√	√	√	
ShuffleWatcher		√	√	
DealyScheduling		√	√	
CoHadoop	√			√
Scarlett		√		√
This Paper	√	√		√

3. Problem of Server Selection

3.1. Task Execution with Data Migration

For the data center with homogeneous servers, all the periodic tasks and online tasks share the resources and data and resources in the cloud data center. Because the tasks we discussed are data-intensive, for each of them, the execution is associated with two important factors: resources and input data. For each server in the cluster, it will be split into uniform resource units like Containers or Virtual Machines (VMs). Therefore, an idle server can allow several tasks to be executed on it at the same time. However, in this paper, to simplify the problem, it is acceptable that we assume each server as one resource slot and each time it could only execute one task. It is easy to expand one server with one resource slot to one server with many resource slots. In addition, the input data is also necessary. The location of data blocks affects the execution of tasks. For each task, there could be two execution modes: locality mode or remote mode. The locality mode indicates the task is executed on the server who hosts the input data for the task. Otherwise, the task executes in remote mode by accessing the input data from another server. It is easy to understand that we always want tasks to be executed in locality mode as much as possible since better data locality means less task execution time. However, the resources is limited and it is impossible to execute all tasks in locality mode.

To evaluate the efficiency of our algorithm, we need to calculate the execution time of all tasks in system. In this paper, the heterogeneous tasks consist of periodic tasks and online tasks. We use P_i to indicate the ith periodic task in the task queue, and O_i represents the ith online task. For the task execution time, we use T_r as the task execution time indicator in remote mode while T_l as the indicator in locality mode.

For the data center, it contains N servers and K different data blocks for task execution. For each data block D_i, it has 3 data replicas deployed on different servers. We use f_i to indicate the required input data of task P_i or O_i. According to the location of f_i when scheduling tasks, we will use the following indicators to classify the task execution mode.

$$\phi(P_i, f_i) = \begin{cases} 0, & \text{locality mode;} \\ 1, & \text{remote mode.} \end{cases} \tag{1}$$

where locality mode means the periodic task P_i is executed locally.

$$\phi(O_i, f_i) = \begin{cases} 0, & \text{locality mode;} \\ 1, & \text{remote mode.} \end{cases} \tag{2}$$

where locality mode means the online task O_i is executed locally.

In our algorithm, we combine delay scheduling with data migration to improve data locality. For the task scheduling decision when some idle resource is available, we need to make a decision by a comparison between the costs of delay scheduling and data migration. With the delay scheduling manner, the selected task will wait until the desired server has idle resource to achieve data locality.

Another method to achieve data locality is to migration the input data from some server to the server with idle resource which will host the selected task. This method is known as data migration in this paper. However, the data migration also bring cost by data transmission. Hence, it is necessary to make a comparison between data migration and delay scheduling. We use T_m to indicate the data migration time. It means the time of transferring a data block from a server to another server is T_m.

3.2. Server Selection

In this paper, we have assumed that each server has one resource slot and can execute one task at each time. Hence, the task scheduling problem can be treated as assign the task to some server or select some server to host the task. Therefore, we can transfer the problem to select a proper server for each task. In a data center, the heterogeneous tasks consist of periodic tasks and online tasks. Periodic tasks are in a queue and wait for scheduling while online tasks arrive in another queue randomly. For a periodic task, it has much longer response time than online task. The goal of scheduling periodic tasks is to shorten the execution time so that improve the efficiency of whole system. While for online tasks, the response time is short. An online task cannot wait for a long time because the users need the system to respond as quickly as possible. To meet this demand, the goal of scheduling online tasks is to scheduling them before the deadline of response. This is more urgent than time reduction. A heterogeneous task scheduling strategy which can satisfy the requirements of periodic tasks and online tasks is necessary.

We use S_j to represent the idle server in data center. As mentioned above, D_i indicates the *i*th data block, and f_i means the replica of D_i. Therefore, we can define the indicators to represent the assignment decisions on idle server S_j as follows.

$$\pi(D_i, S_j) = \begin{cases} 0, & \text{there is no replica of } D_i \text{ on } S_j; \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

$$\pi(P_i, S_j) = \begin{cases} 1, & \text{Periodic task } P_i \text{ is assigned to server } S_j; \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

$$\pi(O_i, S_j) = \begin{cases} 1, & \text{Online task } O_i \text{ is assigned to server } S_j; \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

We want to reduce the task execution time to improve the efficiency in cluster. The running time of whole system is determined by the running time of each server. In addition to the calculation of task execution time, we also need to calculate the running time of servers. We assume that the time is split into time-slots and the workload of each server is used to represent the required running time. Hence, for the workload of some server, say S_j, it contains two parts, the initial workload the tasks assigned the server. The initial workload can be represented by the symbol $L^i(S_j)$. Then, the total workload for server S_j, which also reflects the whole task execution time, could be represented like the following equation:

$$L(S_j) = T(P_i) \cdot \pi(P_i, S_j) + L^i(S_j)$$

Generally, we summarize the related notations in the Table 2, so as to describe the problem properly.

Table 2. Notations.

Symbol	Description
N	number of servers in the data center
K	number of data blocks
$response\_time$	response time of online tasks
$\pi(D_i, S_j)$	data assignment indicator
$\pi(P_i/O_i, S_j)$	task assignment indicator
$\phi(P_i/O_i, f_i)$	task execution mode indicator
$T(P_i/O_i)$	task execution time for task P_i or O_i
$L(S_j)$	workload of server S_j
T_m	data migration time

4. Heterogeneous Task Scheduling

4.1. Scheduling Heterogeneous Tasks with Data Migration

For a data center consists of multiple uniform servers with a default data (blocks) placement, which ensures that there are 3 data replicas for each data. The problem is to assign the heterogeneous tasks on the servers such that the task execution time is minimized. It could be formulated as:

$$min.maxL(S_j), 1 \le j \le N$$
$$s.t. \forall t, \sum_{1}^{K} \pi(D_i, S_j) \le M, 1 \le j \le N$$
$$\forall J \in (\cup O) \cup (\cup P), J.start\_time \le J.response\_time$$

where $L(S_j)$ is the workload, which indicates the task execution time as mentioned above. The number of servers in the data center is N, and the number of data (blocks) is K.

The first constraint means that the hosted data (blocks) for each server should not exceed its storage capacity, represented by M. The second constraint means that the task response time, indicated by $J.response\_time$, should be greater than the task start time, $J.start\_time$, where $\cup O$ means the set of online tasks while $\cup P$ represents the set of periodic tasks, and we have $|\cup O| = m$, $|\cup P| = n$.

Since there are N servers working at the same time, each server has a workload $L(S)$. If we find that server S_j has the maximum workload and we minimize the maximum load of S_j, the whole job execution time of system will be reduced. To achieve this goal, each time we assign a task to the server, we select the server with the least workload in the system, and execute the task in the mode with the least execution time. Obviously, it could be better to achieve data locality for the tasks to reduce the task execution time. Hence, the problem is to pick one task from the queue to occupy the available computing resources. If there is no proper task, which means none of the tasks could be executed locally, we need to make a decision between waiting and remote access to achieve better benefit. Therefore, we propose the heterogeneous task scheduling as shown in Algorithm 1.

Algorithm 1 Heterogeneous task scheduling.

```
Require: $\cup O$: the set of online tasks which have arrived the system;
    $\cup P$: the set of periodic tasks which have arrived the system;
    $S_v$: the server with minimum workload;
    $S_i$: the $i^{th}$ server;
    $t$: the final completion time;
1:  $t \leftarrow 0$;
2:  for $i \leftarrow 1$ to $N$ do
3:    if $S_i.slot > 0$ then
4:      if $\exists O_j \in \cup O \textbf{and} \pi(D_j, S_i) = 1$ then
5:        assign($O_j, S_i$);
6:      else if $\forall O_j \in \cup O, \pi(D_j, S_i) = 0$ then
7:        if $\pi(D_j, S_v) = 1$ then
8:          if $\Delta T = L^i(S_v) - t + T_l(O_j)$ and $L^i(S_v) - t < O_j.response\_time$ then
9:            assign($O_j, S_v$);
10:         else
11:           assign($O_j, S_i$);
12:       else if $\pi(D_j, S_v) = 0$ then
13:         ScheduleTaskwithDataMigration($O_j$);
14:     else if $\cup O = \emptyset \textbf{and} \exists P_j \in \cup P, \pi(D_j, S_i) = 1$ then
15:       assign($P_j, S_i$);
16:     else if $\cup O = \emptyset \textbf{and} \forall P_j \in \cup P, \pi(D_j, S_i) = 0$ then
17:       if $\pi(D_j, S_v) = 1$ then
18:         $\Delta T = min(L^i(S_v) - t + T_l(P_j), T_r(P_j))$;
19:         if $\Delta T = L^i(S_v) - t + T_l(O_j)$ then
20:           assign($P_j, S_v$);
21:         else
22:           assign($P_j, S_i$);
23:       else if $\pi(D_j, S_v) = 0$ then
24:         ScheduleTaskwithDataMigration($P_j$);
25: $t \leftarrow t + 1$;
26: update();
27: return $t$;
```

In Algorithm 1, S_i is an idle server with available resource slots and we need to assign a task to it. Because the heterogeneous tasks consist of periodic tasks and online tasks, we first check whether there is an online task in the queue. If the queue of online tasks is not empty, we will select one task O_j and judge if it can be executed locally. We assign the task O_j to S_i if it can access its input data from S_i. Otherwise, we find the server S_v with minimum workload.if S_v has the required data of task O_j in the data center, we make a comparison between two kinds of time: (1) the waiting time until S_v is idle, and the execution time locally, represented by by $L^i(S_v) - t + T_l(O_j)$; (2) the time of executing task O_j on server S_i in remote mode, the time is represented by $T_r(O_j)$. We compare them and find the less one and indicate it with ΔT. Here, with the result of ΔT, we have two choices as follows:

- $\Delta T = L^i(S_v) - t + T_l(O_j)$. It means we can schedule task O_j to be executed on server S_v so that the remote task can be a local task and reduce the execution time. However, this a an online task scheduling problem. As mentioned before, an online task has a constraint that it has to be executed before its deadline of response time. It means the waiting time until the server S_v to be idle cannot exceed the response time of O_j. Therefore, before we schedule O_j, we should guarantee that the waiting time $L^i(S_v) - t$ is less than $O_j.response\_time$. This manner of scheduling refers to delay scheduling.

- $\Delta T = T_r(O_j)$. If the time of executing task O_j in remote mode is minimum, we can assign task O_j to server S_i and execute it in remote mode immediately.

If task O_j cannot be executed locally on server S_i and there is no local data blocks of task O_j on server S_v, we could consider whether to introduce data migration to execute O_j. The strategy of scheduling tasks with data migration will be interpreted in next subsection.

If the queue of online tasks is empty, we will schedule the periodic tasks. We first check if there is a local periodic task P_j and assign it to the idle server S_i. If all periodic tasks cannot be executed locally on S_i, we need to find the task (say P_j) with the least execution time, indicated as ΔT. Then, we still need to check the execution mode for P_j. If $\Delta T = L^i(S_v) - t + T_l(P_j)$, the task P_j will be assigned to server S_v with locality mode by data migration. Or, the task P_j will be assigned to server S_i with remote execution mode, as shown in the algorithm. Since periodic tasks have much longer response time than online tasks, we do not need consider the waiting time and response time.

4.2. Data Migration

In Section 3 of this paper, we have discussed that data migration will bring extra time consumption. Because data transferring costs network, it must influence the execution of remote tasks. Therefore, we must consider the impacts on remote tasks before migrating data blocks. We explain the process of scheduling a task with data migration in Algorithm 2 as follows.

Algorithm 2 ScheduleTaskwithDataMigration(J).

```
Require: $S_i$: the $i_th$ server; $J$:the selected task; $S_v$:the server with minimum workload in cluster
    without $D_J$.
 1: if There is a remote task on $S_v$ and $(L^i(S_v) - t \leq T_m)$ then
 2:     $\Delta T = minL^i(S_v) - t + T_m + T_l(J), T_r(J)$;
 3: else if There is a remote task on $S_v$ and $(L^i(S_v) - t > T_m)$ then
 4:     $\Delta T = min2(L^i(S_v) - t) + T_m + T_l(J), T_r(J)$;
 5: else if There is a local task on $S_v$ and $(L^i(S_v) - t \leq T_m)$ then
 6:     $\Delta T = minT_m + T_l(J), T_r(J)$;
 7: else if There is a local task on $S_v$ and $(L^i(S_v) - t > T_m)$ then
 8:     $\Delta T = minL^i(S_v) - t + T_l(J), T_r(J)$;
 9: if $\Delta T = T_r(J)$ then
10:     assign($J, S_i$);
11: else
12:     migration($f_J, S_v$);
13:     assign($J, S_v$);
```

In Algorithm 2, task J can be executed on S_i in remote mode or executed on S_v after data migration. Both of the two execution modes need data transmission, which brings transmission cost. We need to compare two time costs: $L^i(S_v) - t$ and T_m. The first $L^i(S_v) - t$ means the task on server S_v still needs to take $L^i(S_v) - t$ to execute. T_m is the time of transferring required data block by selected task J to server S_v. We consider data migration while the S_v is executing a previous task. If T_m is larger than $L^i(S_v) - t$, which means the data migration will not finish when S_v is idle. Hence, task J needs to wait for the end of data transferring before it can be executed. Otherwise, data migration will finish before S_v being idle, and task J can be executed when S_v completes its previous task. Based on this time comparison and decision on if there is a remote task on server S_v, we can list four scenarios as follows:

- There is a remote task on S_v and $(L^i(S_v) - t \leq T_m)$: In this condition, the time of migrating data is over the rest execution time of previous task on S_v, task J needs to wait the end of data migration before it can be executed. Because the previous task on S_v is remote task, $L^i(S_v) - t$ and T_m will double. The work load of S_v will increase $(L^i(S_v) - t)$, so that the new workload of S_v is :

$$L^i(S_v) + (L^i(S_v) - t)$$

Because the previous task on S_v will finish before data migration, the time of migrating data will not be twice when S_v is idle. The rest time for task J to wait and be executed when S_v is idle is:

$$\frac{2T_m - 2(L^i(S_v) - t)}{2} + T_l = T_m - (L^i(S_v) - t) + T_l$$

Therefore, the task execution time with data migration could be:

$$L^i(S_v) + (L^i(S_v) - t) + T_m - (L^i(S_v) - t) + T_l - t = L^i(S_v) - t + T_m + T_l$$

The time of executing task J on S_i in remote mode is $T_r(J)$, we compare these two kinds of time and select the mode with less time.

- There is a remote task on $S_v \&\& (L^i(S_v) - t) > T_m$: In this condition, data migration has finished before S_v is idle. The task J could be executed directly on S_v. $L^i(S_v) - t$ and T_m will also double when executing a remote task and migrating data at the same time on S_v. If data migration is finished, S_v will still take some time to execute its remote task, and the time is:

$$\frac{2(L^i(S_v) - t) - 2T_m}{2} = (L^i(S_v) - t) - T_m$$

The new workload of server S_v is:

$$L^i(S_v) + 2T_m + (L^i(S_v) - t) - T_m$$

Therefore, the whole time for task J to wait and be executed on S_v is:

$$L^i(s_V) + 2T_m + (L^i(S_v) - t) - T_m + T_l - t = 2(L^i(S_v) - t) + T_m + T_l$$

We also compare the time above with $T_r(J)$ and select the less one.

- There is a local task on $S_v \&\& (L^i(S_v) - t) \leq T_m$: Because local tasks do not need network, data migration will not affect local tasks. Similarly, we compare T_m and $L^i(S_v) - t$, and calculate the whole time for task J to be executed on S_v, the time is T_m. We choose the less one between $T_m + T_l(J)$ and $T_r(J)$ and execute J in corresponding mode.
- There is a local task on $S_v \&\& (L^i(S_v) - t) > T_m$: Similar to the situation above, we calculate the whole time for task J to be executed on S_v, the time is $L^i(S_v) - t + T_l(J)$. We compare it with $T_r(J)$ and choose the less one.

From the case analysis, we find that we will execute J on S_i in remote mode with few cases. Otherwise, we migrate the required data of J to from other server to server S_v and execute it in locality mode. Here, we will introduce more about data migration. The algorithm on how to migrate a data block is described in Algorithm 3.

For the migration procedure, we first judge if the destination server has enough storage capacity ($S_v.space > 0$), we can migrate the data block to it and the number of data replica ($f_J.replica$) increases by one. If target server has no space to store the data block, we will find the data block D_d with least degree, which means there is few task needs the data block. If the replica number of D_d is more than one, we can replace it with the migrated data block. This is because we have to guarantee that each data block in the data center must has at least one replica. If the replica is the only one, we give up the operation of data migration.

Algorithm 3 migration(f_J, S_v).

```
Require: $J$: the selected task; $S_v$: the destination server; $num$: the number of replicas of each data;
    $degree$: the number of tasks which require data $f_J$ as input;$S_v$: the target server for data migration.
1: if $S_v.space > 0$ then
2:     assign $f_J$ to $S_v$;
3:     $f_J.replica+ = 1$;
4: else
5:     for $i \leftarrow 1$ to $M$ do
6:         $D_d \leftarrow minimalDegree(S_v)$;
7:         if $D_d.num > 1$ then
8:             replace $f_d$ with $f_J$;
```

5. Experimental Results

To evaluate the proposed approach, we first establish a time model by some real experiments of task execution and data migration. Then, we evaluate our heterogeneous task scheduling algorithm by comparing it with other scheduling approaches.

5.1. Time Model

To validate and establish the time model, we transfer data blocks with different size and research the effect of network competition on time. We test the data transmission time on five groups with various data sizes. For each group, first, we transfer one data block each time from one server to another. Then we transfer two data blocks with different size at the same time. The results are shown in Table 3, where the time unit is millisecond (ms). Learn from results, we can conclude that the network competition will almost doubles the time of transferring time.

Table 3. Time of transferring different data blocks (ms).

Data Size	1 Data Block	2 Data Block
64 MB	947,829	1,519,609
128 MB	1,495,165	2,669,139
256 MB	2,648,865	4,950,684
512 MB	4,939,071	9,511,114
1024 MB	9,535,483	20,151,628

Because HDFS has a default data block size of 64 MB, we ran a sorting program on a server and record the results with and without data migration, which means that the program reads files from local server and from another remote server respectively. We run the program for multiple times and calculate the average value. The final results are shown in Table 4. We analyze the results and find the conclusion as:

$$T_r = \alpha \cdot T_m, \alpha \approx 1.3$$

which means remote tasks spend more time than local tasks, and the remote access time is about 1.3 times of local data access.

Second, data migration will prolong the execution time of remote tasks, but it will not influence local tasks. Our algorithm needs to schedule online tasks and local tasks at the same time. This conclusion tells us that before migrating a data block, we must consider the effects on remote tasks.

Table 4. Tasks executed in two modes (ms).

	With Data Migration	Without Data Migration
Locality mode	1,578,563	1,593,488
Remote mode	3,584,282	2,120,580

5.2. Simulation Analysis

We conduct extensive simulations with various settings and introduce the results on our algorithms. We compare our heterogeneous task scheduling strategy with traditional FIFO strategy and delay scheduling. We also compare it with HRTPS [27], a dynamic scheduling algorithm with precedence constraints for hybrid real-time tasks. The HRTPS classifies the process of task scheduling into static situation and dynamic situation. In static situation, it assigns some of servers to periodic tasks to guarantee that all periodic tasks could be scheduled in a period of time. Then it considers the dynamic situation with real-time tasks arriving randomly. It reserves some servers for these real-time tasks so that they can be scheduled in time. As a result, the HRTPS can schedule heterogeneous tasks simultaneously.

In our experiments, we make the simulation settings as follows. (1) Set $N = 50$ for the data centers, and $M = 20$ for each server; (2) There are $K = 300$ data blocks and 3 replicas for each data block. The replicas are placed in the servers randomly while any server will not host the same replicas; (3) The input data for each task is randomly assigned; (4) The task execution time with data locality is a random value; (5) There are $m = 500$ periodic tasks and $n = 500$ online tasks; (6) The online tasks arrive randomly within 100 time slots; (6) The online tasks must be finished before its deadline, which is a random value within 20 time slots.

We conduct the simulation with different scheduling approaches, the basic results are shown in Figure 1. In the figure, the y-coordinate is the CDF value, which increases by time increasing in x-coordinate. For each point, it means the percentage of completed tasks under the fixed time-slot. The performance of FIFO is not good, and the proposed heterogeneous task scheduling method improves the performance significantly. Meanwhile, the heterogeneous task scheduling method has better performance than HRTPS and delay scheduling.

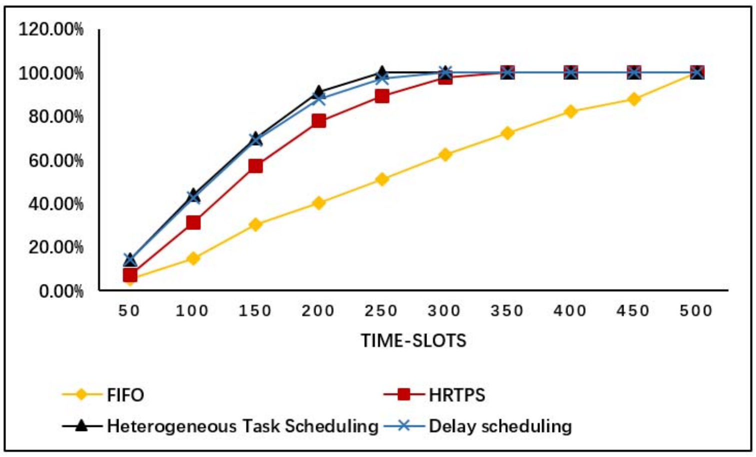

Figure 1. Experimental results of scheduling heterogeneous tasks.

Figure 2 describes the server utilization of each scheduling approach for given time slots. We use $N1$ indicates the number of servers which are occupied. The server utilization can be calculated by $N1/N$ and N is the number of all servers in data center. From the result, we can know that our algorithm has the best server utilization among all three methods at the beginning and decreases from the intermediate period of the experiment, this is because most of tasks have been completed.

The result shows that our heterogeneous task scheduling algorithm has nice performance on server utilization and can reduce the execution time of tasks.

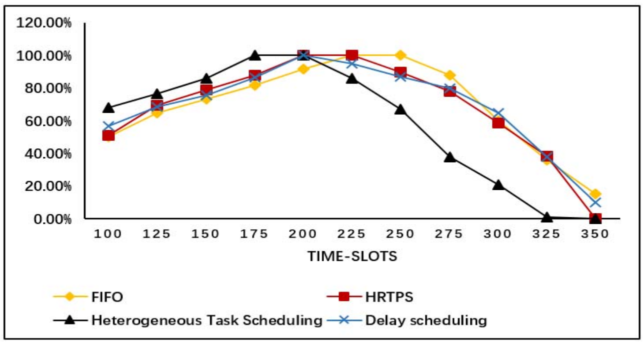

Figure 2. Experimental results of server utilization.

We also conduct experiments to study the differences under different ratios of online tasks to periodic tasks. The number of all tasks are 1000. The results is described in Figure 3. The x-coordinate represents the ratios of online tasks to periodic tasks. The y-coordinate is the time slots which indicate the finish time of all tasks in system. From the results we can conclude that the heterogeneous task scheduling strategy has better performance when the amount of online tasks is large. The result also shows that when the ratio is bigger, the whole execution time is larger. This is because the online tasks will be scheduled to occupy the resources with high priority. However, the limited resource cannot meet the locality task execution for all tasks. Hence, some online tasks are executed in remote mode or locality mode with data migration, both of which brings extra data transmission time. At the same time, the larger ratio means more online tasks are executed without locality mode and the execution time increase.

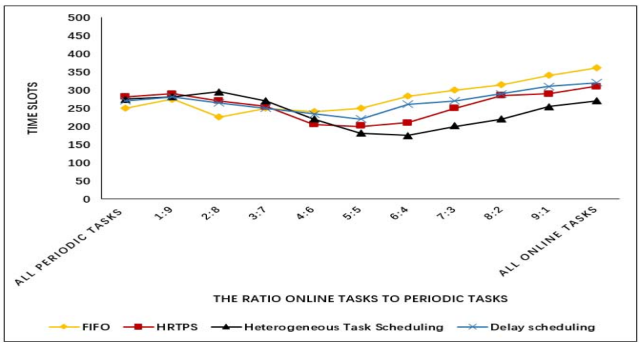

Figure 3. Experimental results of different ratios.

For the above simulations, we also record more data for different approaches. (1) The average server utilization: The server utilization is shown as Figure 2, and we calculate the average server utilization based on the value of each time slot. (2) Overtime online tasks: We record the completion time for each online task and check if it is larger than its deadline. (3) Waiting time: The time duration between the arrival time and scheduled time for the online tasks. We compare these factors with the FIFO, delay scheduling, and HRTPS. The results are shown in Table 5. Learn from the results, we find that our heterogeneous task scheduling algorithm has the least average waiting time, which is the major focus in this paper. For the average server utilization, the value of delay scheduling and HRTPS is close to our algorithm though our algorithm has less task execution time as shown in Figures 1 and 2. This is because the tasks occupy the server with longer time due to remote data access for the delay scheduling and HRTPS algorithm. For the number of overtime online tasks, our algorithm and the delay scheduling algorithm will lead to some overtime cases since the waiting mechanism for tasks. Actually, this could be fixed by adjusting the default waiting time or weight for the online tasks.

Generally, the proposed heterogeneous task scheduling algorithm can improve the performance on task execution time reduction by reasonable data migration. It is a valuable attempt for task scheduling with active manner.

Table 5. Comparison of three methods in three aspects (time slots).

	Averag Server Utilization	**Number of over Time Online Tasks**	**Average Waiting Time**
FIFO	42.3%	5	197
Delay scheduling	64.0%	3	68
HRTPS	68.3%	0	72
Heterogeneous	64.4%	2	64

6. Conclusions

In this paper, we proposed a heterogeneous task scheduling method to schedule periodic tasks and online tasks simultaneously for IoT system. The method take both the delay scheduling and data migration into account to improve data locality. The core idea is to compare the cost of different decisions and choose the reasonable one to schedule the task. We conduct extensive simulations, and the experimental results show that our algorithm has better performance on task execution time reduction compared the FIFO, delay scheduling, and HRTPS. This work indicates that the data migration is an efficient way to improve data locality, which actively occupy resource by moving data from one server to anther server. This is total different from the classical delay scheduling, which waits for idle resource passively. However, it is still a challenging problem to improve the performance of data migration since the system is complex. In addition, it is valuable to discuss how to control the number of data replicas dynamically such that there would be more opportunities for data migration.

Author Contributions: Conceptualization, X.L., L.W. and X.Q.; formal analysis, X.L. and L.W.; funding acquisition, X.L. and I.Y.; investigation, X.L., L.W., J.H.A., X.Q., G.P. and I.Y.; methodology, X.L. and I.Y.; project administration, X.L. and I.Y.; software, X.L. and W.L.; supervision, J.H.A., X.Q. and I.Y.; writing—original draft, X.L., L.W. and J.H.A.; writing—review & editing, X.L., X.Q., G.P. and I.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under Grant 61802182 as well as in part by the Soonchunhyang University Research Fund.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yu, B.; Pan, J. Location-aware associated data placement for geo-distributed data-intensive applications. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon, Hong Kong, China, 26 April–1 May 2015; pp. 603–611.
2. Shi, W.; Gao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
3. Zhu, C.; Zhou, H.; Leung, V.C.M.; Wang, K.; Zhang, Y.; Yang, L.T. Toward big data in green city. *IEEE Commun. Mag.* **2017**, *55*, 14–18. [CrossRef]
4. Li, X.; Tatebe, O. Data-Aware Task Dispatching for Batch Queuing System. *IEEE Syst. J.* **2017**, *11*, 889–897. [CrossRef]
5. Li, D.; Wu, J.; Chang, W. Efficient Cloudlet Deployment: Local Cooperation and Regional Proxy. In Proceedings of the International Conference on Computing, Networking and Communications, Maui, HI, USA, 5–8 March 2018; pp. 757–761.
6. Zaharia, M.; Borthakur, D.; Sarma, J.S.; Elmeleegy, K.; Shenker, S.; Stoica, I. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In Proceedings of the 5th European Conference on Computer Systems, Paris, France, 13–16 April 2010; ACM: New York, NY, USA, 2010; pp. 265–278.
7. Li, X.; Wang, L.; Lian, Z.; Qin, X. Migration-Based Online CPSCN Big Data Analysis in Data Centers. *IEEE Access* **2018**, *6*, 19270–19277. [CrossRef]

8. Sharma, V.; You, I.; Kumar, R. Energy Efficient Data Dissemination in Multi-UAV Coordinated Wireless Sensor Networks. *Mob. Inf. Syst.* **2016**, *2016*, 8475820:1–8475820:13. [CrossRef]
9. Sun, R.; Yuan, J.; You, I.; Shan, X.; Ren, Y. Energy-aware weighted graph based dynamic topology control algorithm. *Simul. Model. Pract. Theory* **2011**, *19*, 1773–1781. doi:10.1016/j.simpat.2010.09.002. [CrossRef]
10. Deng, Y.; Chen, Z.; Zhang, D.; Zhao, M. Workload scheduling toward worst-case delay and optimal utility for single-hop Fog-IoT architecture. *IET Commun.* **2018**, *12*, 2164–2173. [CrossRef]
11. Chen, M.; Hao, Y. Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 587–597. [CrossRef]
12. Zhang, W.; Zhang, Z.; Zeadally, S.; Chao, H. Efficient Task Scheduling with Stochastic Delay Cost in Mobile Edge Computing. *IEEE Commun. Lett.* **2019**, *23*, 4–7. [CrossRef]
13. Wang, W.; Zhu, K.; Ying, L.; Tan, J.; Zhang, L. Map task scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Trans. Netw.* **2016**, *24*, 190–203. [CrossRef]
14. Dean, J.; Ghemawat, S. MapReduce: A flexible data processing tool. *Commun. ACM* **2010**, *53*, 72–77. [CrossRef]
15. Apache Hadoop. Available online: http://hadoop.apache.org/ (accessed on 15 June 2020) .
16. Thomas, L.; Syama, R. Survey on MapReduce scheduling algorithms. *Int. J. Comput. Appl.* **2014**, *95*, 9–13. [CrossRef]
17. Ahmad, F.; Chakradhar, S.T.; Raghunathan, A.; Vijaykumar, T.N. Shufflewatcher: Shuffle-aware Scheduling in Multi-tenant Mapreduce Clusters. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIXATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 1–12.
18. Jin, J.; Luo, J.; Song, A.; Dong, F.; Xiong, R. BAR: An Efficient Data Locality Driven Task Scheduling Algorithm for Cloud Computing. In Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach, CA, USA, 23–26 May 2011; pp. 295–304.
19. Lee, Y.C.; Zomaya, A.Y. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *22*, 1374–1381. [CrossRef]
20. Chen, Q.; Zhang, D.; Guo, M.; Deng, Q.; Guo, S. SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In Proceedings of the IEEE International Conference on Computer and Information Technology, Bradford, UK, 29 June–1 July 2010; pp. 2736–2743.
21. Eltabakh, M.Y.; Tian, Y.; Ozcan, F.; Gemulla, R.; Krettek, A.; McPherson, J. CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop. *Proce. VLDB Endow* **2011**, *4*, 575–585.
22. Ananthanarayanan, G.; Agarwal, S.; Kandula, S.; Greenberg, A.G.; Stoica, I.; Harlan, D.; Harris, E. Scarlett: Coping with Skewed Content Popularity in Mapreduce Clusters. In Proceedings of the European Conference on Computer Systems, Proceedings of the Sixth European Conference on Computer systems (EuroSys 2011), Alzburg, Austria, 10–13 April 2011; pp. 287–300.
23. Ananthanarayanan, G.; Ghodsi, A.; Warfield, A.; Borthakur, D.; Kandula, S.; Shenker, S.; Stoica, I. PACMan: Coordinated Memory Caching for Parallel Jobs. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, San Jose, CA, USA, 25–27 April 2012; pp. 267–280.
24. Li, X.; Wu, J.; Qian, Z.; Tang, S.; Lu, S. Towards location-aware joint job and data assignment in cloud data centers with NVM. In Proceedings of the 36th IEEE International Performance Computing and Communications Conference (IPCCC 2017), San Diego, CA, USA, 10–12 December 2017; IEEE Computer Society: Washington, WA, USA, 2017; pp. 1–8.
25. Li, C.; Tang, J.; Tang, H.; Luo, Y. Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment. *Future Gener. Comput. Syst.* **2019**, *95*, 249–264. [CrossRef]
26. Zhou, J.; Sun, J.; Cong, P.; Liu, Z.; Zhou, X.; Wang, T.; Wei, T.; Hu, S. Security-Critical Energy-Aware Task Scheduling for Heterogeneous Real-Time MPSoCs in IoT. *IEEE Trans. Serv. Comput.* **2020**, *13*, 745–758. [CrossRef]
27. Yin, J.; Gu, G.; Zhao, J. Dynamic scheduling algorithm for hybrid real-time tasks with precedence constraints. *Comput. Integr. Ated. Manuf. Syst.* **2010**, *16*, 411–416.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).