Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 2102348, 11 pages
https://doi.org/10.1155/2018/2102348

WILEY | Hindawi

Research Article

A Task Scheduling Algorithm Based on Classification Mining in Fog Computing Environment

Lindong Liu,[1,2] **Deyu Qi**,[1] **Naqin Zhou**,[3] **and Yilin Wu**[2]

[1]*Research Institute of Computer Systems, South China University of Technology, Guangzhou, China*
[2]*Department of Computer Science, Guangdong University of Education, Guangzhou, China*
[3]*Cyberspace Institute of Advanced technology, Guangzhou University, Guangzhou, China*

Correspondence should be addressed to Lindong Liu; hongox@163.com

Received 28 April 2018; Revised 18 June 2018; Accepted 5 July 2018; Published 1 August 2018

Academic Editor: Fuhong Lin

Fog computing (FC) is an emerging paradigm that extends computation, communication, and storage facilities towards the edge of a network. In this heterogeneous and distributed environment, resource allocation is very important. Hence, scheduling will be a challenge to increase productivity and allocate resources appropriately to the tasks. We schedule tasks in fog computing devices based on classification data mining technique. A key contribution is that a novel classification mining algorithm I-Apriori is proposed based on the Apriori algorithm. Another contribution is that we propose a novel task scheduling model and a TSFC (Task Scheduling in Fog Computing) algorithm based on the I-Apriori algorithm. Association rules generated by the I-Apriori algorithm are combined with the minimum completion time of every task in the task set. Furthermore, the task with the minimum completion time is selected to be executed at the fog node with the minimum completion time. We finally evaluate the performance of I-Apriori and TSFC algorithm through experimental simulations. The experimental results show that TSFC algorithm has better performance on reducing the total execution time of tasks and average waiting time.

1. Introduction

Many applications, such as health monitoring application or intelligent traffic control application may need to receive feedback in a short amount of time, and the latency due to sending data to the cloud and then returning the response from the cloud to the operator of these programs has bad effects [1]. So, in 2012, Bonomi presented a novel concept called the fog computing [2]. Fog computing consists of a large number of geographically distributed fog servers which can be cellular base stations, access points, gateways, switches, and routers with limited capabilities, as compared to specialized computing facilities such as data centers [3–5]. In fog computing, the massive data generated by different kinds of Internet of Things (IoT) [6, 7] devices can be processed at the network edge instead of transmitting it to the centralized cloud infrastructure due to bandwidth and energy consumption concerns [8]. Fog computing has become a new computing model in providing local computing resources and storage for end-users rather than cloud computing.

The contradiction [9, 10] between computation intensive applications and resource limited devices becomes the bottleneck for providing satisfactory quality of experience. This contradiction needs to be solved by task scheduling in fog computing environment. Task scheduling is widely applied in distributed computing systems and the cloud computing environment [11, 12]. Task scheduling in fog computing is to allocate appropriate resources for application tasks. How to select appropriate resources for the application task to meet the minimum completion time, to satisfy the users' quality of service (QoS) requirements, to improve the fog computing throughput, and to achieve the load balancing scheduling can be defined as task scheduling problem in fog computing environment. Therefore, it is of great practical significance to achieve efficient resource utilization and higher performance in the fog computing environment.

In fog computing environment, task scheduling depends on whether there are dependencies between the tasks that are scheduled. It can be divided into independent task scheduling

and related task scheduling. Related task scheduling is often referred to as dependent task scheduling [13]. There is no dependency relationship and data communication among tasks in independent task scheduling [14, 15]. Dependent task scheduling has some dependence and there is data communication among tasks. A typical task scheduling model is built on the basis of graphs, usually called task graphs. The most common task graph is Directed Acyclic Graph (DAG), so the dependent task scheduling is also called DAG scheduling.

Before tasks are scheduled, tasks have two ways to arrive. One is the batch mode. When all tasks arrive, they are allocated to the corresponding fog nodes through a scheduling algorithm. Another is the online mode. The arrival time of each task is random and a task is scheduled to a fog node as soon as it arrives at the RMS (resource management system). Task scheduling of fog nodes has been proved to be a NP-complete problem [16]. The research work of task scheduling is a very important aspect and has been widely and deeply studied by researchers [17]. At present, although many research achievements have been obtained for task scheduling, researchers are still continuing to explore and study [18]. Research of scheduling tasks in fog computing environment has not been well-established yet due to the lack of fog architecture that manages and allocates resources efficiently. Our research also has a positive influence on some optimization problems [19–22].

The rest of the paper is organized as follows. In Section 2 we describe the related work of the research. In Section 3, we introduce the classification mining algorithm and an improved I-Apriori algorithm. In Section 4, we introduce a task scheduling model, the scheduling algorithm, and the scheduling process in fog computing. The analysis of the experimental process and experimental results of task scheduling algorithm are given in Section 5, followed by our conclusion made in Section 6.

2. Related Work

2.1. Related Work of Classification Mining. Classification mining algorithms are widely used in text, image, video, traffic, medical, big data, and other application scenarios. A pipelined architecture for the implementation of axis parallel binary DTC was proposed in [23] that dramatically improves the execution time of the algorithm while consuming minimal resources in terms of area. Reference [24] proposed a fast and accurate data classification approach which can learn classification rules from a possibly small set of records that are already classified. The proposed approach is based on the framework of the so-called Logical Analysis of Data (LAD). The accuracy and stability of the proposed algorithm are better than that of the standard LAD algorithm. Sequence classification was introduced in [25] using rules composed of interesting patterns found in a dataset of labelled sequences and accompanying class labels. They measure the interestingness of a pattern in a given class of sequences by combining the cohesion and the support of the pattern. They use the discovered patterns to generate confident classification rules and present two different ways of building a classifier. The patterns that the algorithm discovers represent the sequences well and are proved to be more effective for the classification tasks than other machine learning algorithms. A Bayesian classification approach for automatic text categorization using class-specific features was proposed in [26]. Unlike conventional text categorization approaches, the method selects a specific feature subset for each class. One noticeable significance of the algorithm is that most feature selection criteria such as Information Gain (IG) and Maximum Discrimination (MD) can be easily incorporated into the algorithm. Compared with other algorithms, it demonstrates that the algorithm is effective and further indicates its wide potential applications in data mining. Furthermore, we will apply this algorithm to other areas, such as oblivious RAM [27, 28], string mapping [29], and match problem [30].

2.2. Related Work of Independent Task Scheduling. For a large scale environment, e.g., cloud computing system, there had been also numerous scheduling approaches proposed with the goal of achieving the better task execution time for cloud resources [31]. Independent task scheduling algorithms mainly include MCT algorithm [32], MET algorithm [32], MIN-MIN algorithm [33], MAX-MIN algorithm [33], PMM algorithm, and genetic algorithm. The MCT (Minimum Completion Time) algorithm assigns each task in any order to the processor core that causes the task to be finished at the earliest time. It makes some tasks unable to be allocated to the fastest processor core. The MET (Minimum Execution Time) algorithm assigns each task to a processor core in any order that minimizes the execution time of the task. Contrary to the MCT algorithm, the MET algorithm does not consider the processor core's ready time, which may lead to serious load imbalance across processor cores. The MIN-MIN algorithm calculates the minimum completion time of all unscheduled tasks firstly, and then selects the task with the minimum completion time and assigns the task to the processor core that can minimize its completion time, repeating the process many times until all tasks are scheduled. The same as the MCT algorithm, the MIN-MIN algorithm is also based on the minimum completion time. The MIN-MIN algorithm considers all tasks that are not scheduled, but the MCT algorithm considers only one task at a time. The MAX-MIN algorithm is similar to the MIN-MIN algorithm, which also calculates minimum completion time without scheduled tasks firstly and then selects the task with the largest minimum completion time and assigns the task to the processor core with the minimum completion time. The PMM (Priority MIN-MIN) algorithm is an improvement of the MIN-MIN algorithm. It does not choose the smallest task with the earliest complete time, but it selects k tasks with smaller earliest completion time and schedules the task with highest priority in the k tasks. The PMM algorithm takes the standard deviation of the task on each processor core as the priority of the task. The higher the standard deviation, the higher the task priority.

On one hand, literature of existing classification algorithms applies decision tree algorithm and Bayes classification algorithm to various application scenarios. On the other hand, combined with cloud computing, distributed computing, big data, grammatical evolution [34, 35], and other

technologies, researchers are focused on how to optimize and improve the performance of classification algorithms. In task scheduling, few researchers apply the classification mining algorithm to schedule tasks.

3. Classification Data Mining

3.1. Overview. Classification mining algorithm [36] is the key technology of data mining. As a supervised learning algorithm, it is based on existing training data sets to set up a model to predict the categories of new data sets. It can find classification rules and predict new data types through analysis of the training data set. A classification mining algorithm consists of two stages which are building the model phase and using the model phase. In the first stage, it analyzes the existing training data set and builds a corresponding model and then generates some classification rules. In the second stage, it classifies new data sets based on the constructed classification model.

Major classification mining algorithms include random decision forests [37], decision tree algorithm, Bayes algorithm, genetic algorithm, artificial neural network algorithm [34], and classification algorithm based on association rules. Classification algorithm is widely used in wireless sensor networks, network intrusion detection, call logs, and risk assessment in banks. In this paper, the classification algorithm based on association rules is introduced and the Apriori algorithm is improved and evaluated.

3.2. Mining Model. Apriori [35, 38, 39] is a classical classification algorithm based on association rules (CBA). It generates frequent itemsets through an iterative process. The Apriori algorithm includes two steps. First of all, it finds frequent itemsets from a known transaction in which the frequency is greater than or equal to minimum support threshold through pruning and connection operation of frequent itemsets. Then, it generates association rules based on the frequent itemsets and minimum confidence degree.

The improved association rule mining model is implemented in two steps. (1) Firstly, the transaction database D is scanned to store the transaction identification *TID* for each itemset, and the candidate 1-itemset C_1 is generated. Delete the itemsets from C_1 which are less than the minimum support threshold, and get the frequent 1-itemsets of L_1. (2) Loop execution of the process is done until $L_{k\text{-}1}$ is empty. Firstly, let $L_{k\text{-}1}$ and $L_{k\text{-}1}$ be joined to generate candidate itemset C_k. Secondly, a new transaction identifier list can be obtained through the intersection of the transaction identifier list, and the count of the itemsets can be obtained directly through C_k. Thirdly, comparing the count of C_k with the minimum support threshold *min_sup*, reserve itemsets which are more than or equal to minimum support threshold *min_sup*, and delete the rest of itemsets; then the final frequent itemset L is generated.

3.3. Improved Association Rule Mining Algorithm. In the process of producing frequent itemsets in the Apriori algorithm, there are two factors that affect the performance of the algorithm. Firstly, it needs to scan the original transaction database every time to generate the frequent k-itemsets, so the number of scanned transaction databases is too much, which can result in the decline of algorithm performance. Secondly, in the process of tree cutting, the algorithm needs to scan candidate k-1 sets to get candidate itemset. Therefore, the algorithm scans itemsets many times; it also leads to the decline of algorithm performance. In view of the above problems, we improve the process of frequent itemsets in the algorithm, and an improved I-Apriori algorithm is proposed based on the Apriori algorithm. The I-Apriori algorithm is described as follows in Algorithm 1.

```
1 Input: transaction database D; min_sup
2 Output: frequent itemsets L
3 C1=find_candidate_1-itemsets(D);
4 int count=the number of TID in D;
5 for each itemset s of C1{
6    s.item-set=s;
7    s.count=count of s in C1;
8 s.tid-list=the set of all TID includes s;
9 if s.count<min_sup*count
10    delete s in C1;
11 }
12 L1=C1;
13 for (k=2; Lk-1 ≠ ∅; k++){
14    for each itemset l1 in Lk-1{
15       for each itemset l2 in Lk-1{
16          c=l1⋈l2;
17          c.tid-list= l1.tid-list∧l2.tid-list;
18          c.count=count TID in c.tid-list;
19       }
20    }
21 if c.count>=min_sup*count
22    add c to Ck;
23    Lk=Ck;
24 }
```

Algorithm 1: I-Apriori algorithm.

In the I-Apriori algorithm, during the process of generating the candidate itemset C_k every time, except for storing the itemset and the count of support degree, it is more important to store the transaction identifier list attribute *Tid-list*. After completing the connection operation between itemsets, the algorithm can get the list of transaction identifiers and the count of itemsets directly through the attribute *Tid-list* and does not need to scan the transaction database again. Based on the above reasons, I-Apriori algorithm can improve the performance effectively.

3.4. Algorithm Evaluation. The efficiency of Apriori algorithm and I-Apriori algorithm is evaluated based on time complexity and algorithm execution time.

3.4.1. Time Complexity. Suppose the number of transactions and items in the transaction database D is n and m, and the iteration times of frequent itemsets in the algorithm is k. The time complexity of classical Apriori algorithm is composed of three layers nested for loops, *apriori_gen* subroutine and

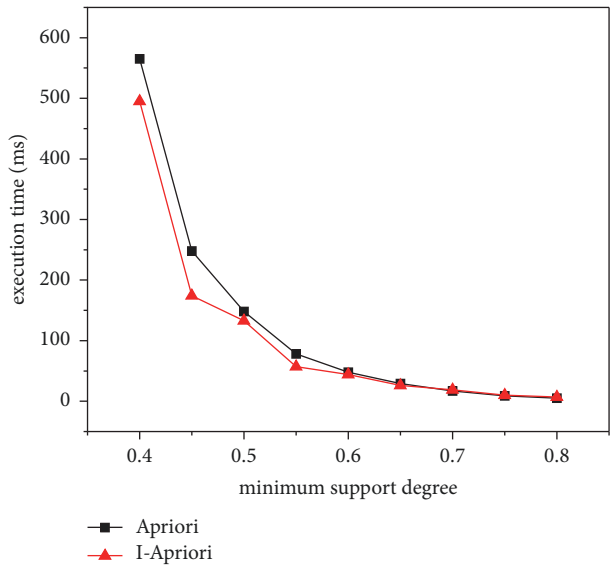


Figure 1: Comparison of execution time.

has_infrequent_subset subroutine called *apriori_gen* subroutine in the main algorithm. It is easy to find that the time complexity of Apriori algorithm is $O(k^4 * m*n)$. According to the I-Apriori algorithm shown in Algorithm 1, because only one time is needed to scan the transaction database D, the time complexity of I-Apriori algorithm is $O(m+n+k^3)$. Obviously, $O(m+n+k^3)$ is better than $O(k^4 * m*n)$. The greater the transaction database D, the more the number of items, the more iterations, and the higher efficiency of the I-Apriori algorithm.

3.4.2. Experimental Analysis. The Java language is used to realize the classic Apriori algorithm and the I-Apriori algorithm, respectively. The hardware environment is Intel 2.5 GHz CPU, 4 GB memory, and the operation system is Windows 7. We generated corresponding frequent itemsets for the transaction database.

When the number of transactions in the transaction database is 200 and the number of items is 20, the execution time needed for the two algorithms to generate frequent itemsets under different minimum support degree (0.4~0.8) is shown in Figure 1. When the number of items in the transaction database is 20 and the minimum support degree is 0.4 and 0.6 (several experiments show that the execution time of the algorithms is longer when the minimum support degree is 0.4, while the algorithm has a shorter execution time when the minimum support degree is 0.6; therefore, 0.4 and 0.6 are chosen to compare the execution time of the two algorithms under different transaction numbers), the execution time needed for the two algorithms to generate frequent itemsets under different number of transactions (50~400) is shown in Figure 2.

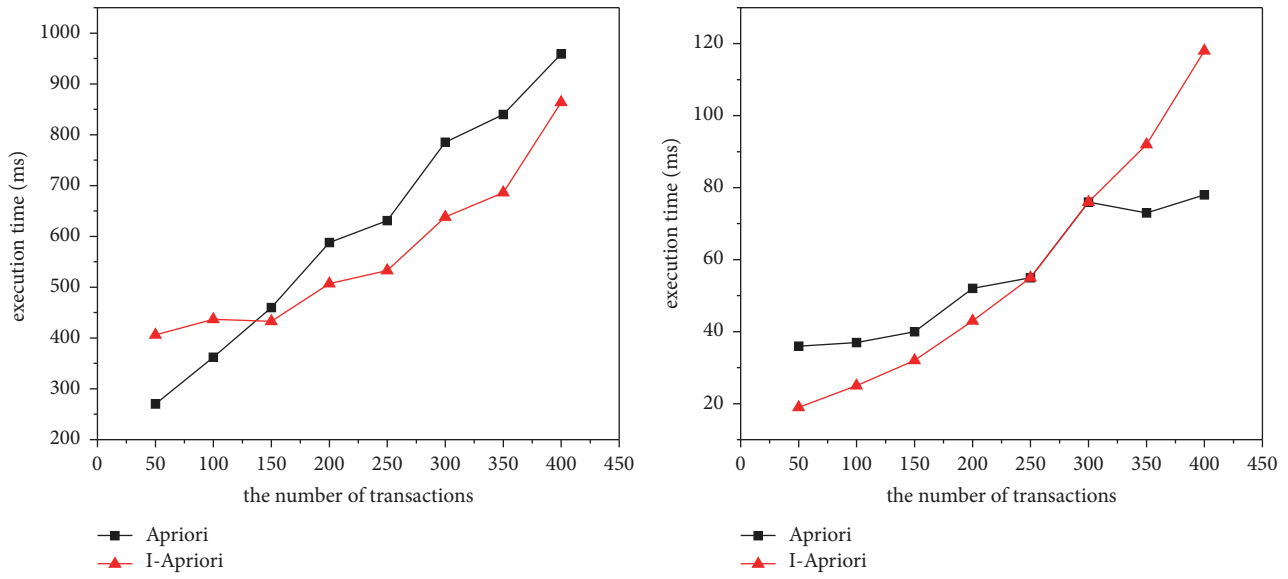
From Figure 1, when the minimum support degree of Apriori algorithm and I-Apriori algorithm is small, the execution time of generating frequent itemsets of I-Apriori algorithm is smaller than that of Apriori algorithm. With the increase of minimum support degree, there is little difference in execution time of the two algorithms. When the minimum support degree is large, the execution time of generating frequent itemsets of I-Apriori algorithm is larger than that of Apriori algorithm. When the minimum support degree is small and the number of iterations is greater, the efficiency of the I-Apriori algorithm is higher. When the minimum support degree is large and the number of iterations is smaller, the efficiency of the Apriori algorithm is higher. Therefore, the I-Apriori algorithm is suitable for smaller minimum support degree and more iterations in classification mining. When the minimum support degree is small, the number of iterations of classification mining will increase. The I-Apriori algorithm will reduce the times of scanning the transaction database significantly, and the execution time of the algorithm is shorter. On the contrary, when the minimum support degree is large, the number of iterations will be decreased. Although the I-Apriori algorithm also can reduce the times of scanning the transaction database, I-Apriori algorithm has no advantage over Apriori algorithm.

In the case of smaller minimum support degree in Figure 3, when the number of transactions is smaller, the execution time of generating frequent itemsets of the Apriori algorithm is smaller than that of the I-Apriori algorithm. With the increase of the number of transactions, the efficiency of I-Apriori algorithm is obviously higher than that of Apriori algorithm. In the case of larger minimum support degree in Figure 4, the execution time of generating frequent itemsets of the Apriori algorithm is larger than that of the I-Apriori algorithm when the number of transactions is small. With the increase of the number of transactions, the Apriori algorithm is more efficient than the I-Apriori algorithm. Generally speaking, the I-Apriori algorithm is suitable for small minimum support degree and large number of transactions when generating frequent itemsets.

4. Task Scheduling of Fog Computing

Task scheduling of fog computing is to schedule tasks to fog nodes with different computing powers, and arrange their execution order reasonably, so that the total execution time is shortest. All notations utilized in the paper are listed in Table 1.

4.1. Fog Computing System Architecture. Fog computing system [40] has three tiers in a hierarchy network, as represented in Figure 3. The front-end tier consists of IoT devices, which serve as user interfaces that send requests from users via WiFi access points or Internet. IoT devices are always subject to strict constraints on their resource such as CPU, memory, and, when run, a very complex application. The fog tier, which is formed by a set of near-end fog nodes, receives and processes part of a workload of users' request. The fog tier is generally deployed near IOT terminals, which provides limited computing resources for users. Users can access the computing resources in the fog tier directly, so it can avoid additional communication delays. The cloud tier consists of multiple servers or cloud nodes. The remote cloud can provide abundant computing resources, but it is located



(a) Execution time under the minimum support degree 0.4

(b) Execution time under the minimum support degree 0.6

FIGURE 2: (a) Comparison of execution time under the minimum support degree 0.4. (b) Comparison of execution time under the minimum support degree 0.6.

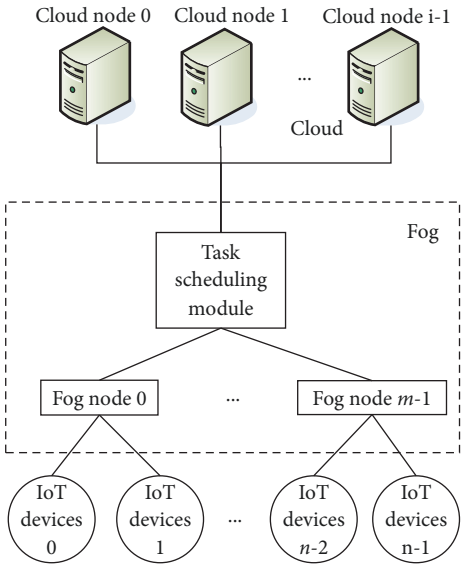


FIGURE 3: Fog computing system architecture.

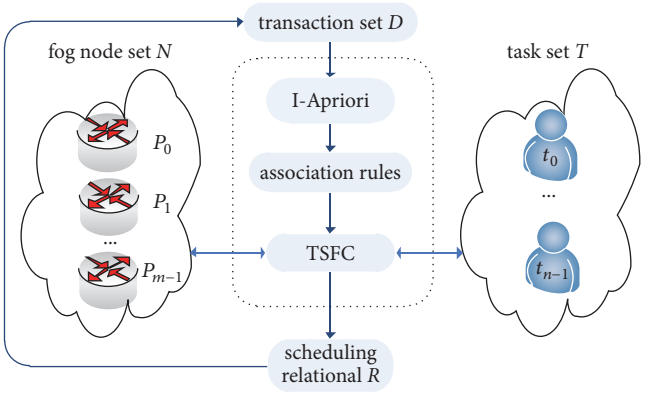


FIGURE 4: Task scheduling model of fog computing.

physically far from the users and the transmission delay is large.

4.2. Task Scheduling Model. In order to implement the task scheduling of fog computing effectively, the classification algorithm is integrated into the task scheduling process of fog computing. Figure 4 presents the task scheduling model of fog computing. In order to realize an effective scheduling process between the fog node set N and the task set T, the scheduling module consists of two algorithms, i.e., I-Apriori algorithm and TSFC (Task Scheduling in Fog Computing) algorithm. Firstly, based on the scheduling transaction set D, association rules of the node set and the task set are generated by the I-Apriori algorithm. Secondly, the association rules are used as the input of TSFC algorithm to get the task scheduling relationship between the fog node set and the task set. Finally, the task scheduling relationship R is inserted into the scheduling transaction set D to provide input data for the next task scheduling.

4.3. TSFC Scheduling Algorithm. Based on the I-Apriori algorithm, TSFC algorithm is designed and is shown in Algorithm 2. The basic idea of the algorithm is to schedule tasks in the task scheduling relational table with higher priority. Set the completion time of these tasks in the table to a larger value, and then select the fog node with the minimum completion time. Execute a loop from the rest of

Table 1: Summary of the notations.

var	definition
$TS(k)$	TS contains k task sets that needs to be scheduled
T	$T=\{t_1,t_2,\cdots,t_n\}$ is a set of n tasks. In this set, all of the tasks in T are independent tasks
N	$N=\{P_0, P_1, \ldots, P_{m-1}\}$ denotes the set of processors
D	An edge $d_{ij} \in D$ denotes a link between processor P_i and P_j
bw_{ij}	the bandwidth between processor P_i and P_j
P	$P=(N,D)$ denotes the topology of a fog computing network
$Time[n,m]$	$Time[i,j]$ descripts the estimated running time of the task t_i on fog node P_j
$D[z]$	Scheduling transaction set contains z transactions
$R[t,c]$	Task scheduling relationship contains the scheduling relationship between the task set T and the fog node N
ST_i	The start execution time of task t_i
AT_i	Actual arrival time of task t_i
FT_i	the completion time of the task t_i
TST	Total task scheduling execution time is the maximum value of all tasks' completion time
AWT	Average waiting time is TST divided by n, $AWT=\sum_{j=1}^{n}(ST_j - AT_j)/n$

```
1 Input: Time[n, m], R[t, c], TS
2 Output: TST, AWT
3 double ST[i], FT[i], min_FT, Total_Time, Time, TST, AWT, WT;
4 int Total_node, Total_Task, Task, Total_TaskSet, best_node_ID;
5 read every taskset from TS;
6 for (i=0; i<Total_TaskSet; i++){
7    read a task from taskset;
8    for (j=0; j<Task; j++){
9       for (k=0; k<Total_node; k++){
10           if (R[j,k]=-1)
11             FT[j]=Time[j,k];
12         else
13             FT[j]=ST[j]-R[j,k];
14       }
15    find minimal min_FT and corresponding k in FT[j] with task j;
16    ST[k]=ST[k]+min_FT;
17    sort all tasks in taskset i from small to large according to FT[j];
18    ST[ ]=0; FT[ ]=0;
19    }
20   while (taskset i is not empty) {
21    select task t_Task with the largest FT[j] in taskset;
22    if t_Task=null
23       break;
24    select the node with smallest FT[j] of t_Task and return best_node_ID;
25    WT=WT+ST[best_node_ID];
26    ST[best_node_ID]=FT[best_nodee_ID];
27    delete t_Task;
28    Task=Task-1;
29    recalculate every FT[j] of the rest of tasks;
30    sort all tasks in taskset i from small to large according to FT[j];
31    }
32   write the largest ST[j] to Time of all tasks;
33   Total_Time=Total_Time+Time;
34   Total_Task=Total_Task+Task;
35   TST=Total_Time;
36   AWT=WT/Total_Task;
37 }
```

Algorithm 2: TSFC algorithm.

Table 2: Execution time matrix Time$[n,m]$ of task set T and fog node set N.

task	P_0	P_1	P_2	P_3
t_0	200	211	180	223
t_1	102	122	91	130
t_2	81	92	88	95
t_3	55	59	57	61
t_4	32	33	29	36
t_5	155	160	149	173
t_6	287	291	267	305
t_7	135	142	122	160
t_8	228	237	204	251
t_9	178	183	161	195

the tasks to select the task with minimum completion time to schedule and assign the selected task to the fog node with minimum completion time until all of the tasks are scheduled. Supposing the number of task sets, tasks, and fog nodes is k, n, and m, respectively, the time complexity of TSFC algorithm is $O(k*n^2+k*n*m)$.

4.4. Analysis of Scheduling Process. In order to understand and analyze the TSFC algorithm, a complete case is used to analyze the scheduling process of the TSFC algorithm. We analyze the whole process of task scheduling algorithm of fog computing. Suppose that the task set T contains 10 tasks and the node set C includes 4 fog nodes; that is, n=10 and m=4. The execution time matrix $Time[n,m]$ of task set T and node set C is shown in Table 2.

(1) Transaction database. Transaction set $D[z]$ is shown in Table 3. Each scheduling information between the task set T and the fog node set N is stored as a transaction information. A Boolean value is used to describe whether the task or node is scheduled or not. The Boolean true value representing the task or node is scheduled. On the contrary, the Boolean false value representing the task or node is not scheduled. In addition, it is assumed that the transaction set D contains 10 transactions; that is, z=10.

(2) Classification mining. The transaction database D is used as the input of I-Apriori algorithm, and the minimum support degree min_sup=0.5. Frequent itemsets $\{P_1,P_0,P_3,t_7\}$ and $\{P_2,P_0,P_3,t_7\}$ are generated by the I-Apriori algorithm. Association rules $t_7 => P_1 \vee P_0 \vee P_3$ (minimum confidence degree is equal to 0.833) and $t_7 => P_2 \vee P_0 \vee P_3$ (confidence degree is equal to 1.0) are generated with the minimum confidence degree min_conf=0.8.

(3) Task scheduling relational table. According to the association rules generated by the I-Apriori algorithm, the scheduling relationship between the task set and fog node set is shown in Table 4. In the task scheduling relational table $R[t,N]$, there are three kinds of values of task t_i corresponding to fog node P_j. In the first case, if the task t_i and the fog node P_j do not appear in the association rules, every value of the row corresponding to task t_i is equal to −1. In the second case, if task t_i and fog node P_j appear in the association rules, then calculate the confidence degree of task t_i on the fog node P_j. Let the confidence degree of tsk t_i corresponding to each fog node P_j be $tP_k(k\in[1,m])$, and the value of task t_i and fog node P_j is equal to $tP_k/\sum_{k=0}^{m-1} tP_k$ in the task scheduling relational table $R[t,N]$. For example, the scheduling relationship value between t_7 and P_1 is $0.833/(0.833 + 0.833 + 0.833 + 1.0 + 1.0 + 1.0)*100$=11.15. In the last case, the corresponding scheduling relationship value is equal to 0 when the fog node does not appear in the association rules.

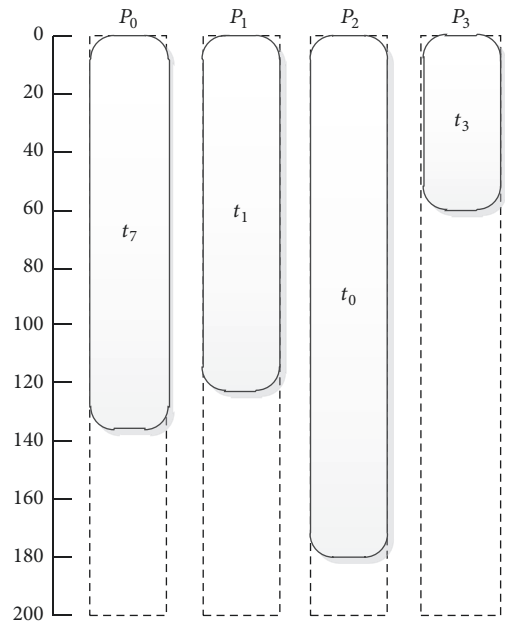


Figure 5: Scheduling diagram between tasks and fog node.

(4) Scheduled tasks TS. Scheduled tasks TS is a task list that needs to be scheduled in an experiment. Suppose the arrival time (AT_i) of all tasks is equal to 0. The task set to be scheduled is shown in Table 5.

(5) Task scheduling. Because all of the tasks are independent, the communication cost among tasks is not considered in TSFC algorithm. The value of every element of the communication matrix is equal to 0. The task set is scheduled based on the TSFC algorithm with Tables 2, 4, and 5 as input. Then, output the execution time of (TST) and the average waiting time (AWT) of the scheduled tasks.

Take the first task set$\{t_0,t_1,t_3,t_7\}$ in the scheduled tasks TS as an example. Task t_7 is scheduled to fog node firstly because the task t_7 appears in the association rules, and task t_7 is scheduled on fog node P_0 or P_3. Recalculate the minimum completion time of the three tasks $\{t_0,t_1,t_3\}$ in the task set 1, and select task t_0 with the largest minimum completion time to be scheduled on fog node P_2. Next, recalculate the minimum completion time of the remaining two tasks $\{t_1,t_3\}$ again, and task t_1 is scheduled on fog node P_1. Finally, task t_3 is scheduled on fog node P_3. The scheduling relationship between the task and fog node in the task set 1 is shown in Figure 5.

Table 3: Transaction database.

transaction	P_0	P_1	P_2	P_3	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
T_1	T	T	T	T	T	F	T	F	F	T	F	T	F	F
T_2	F	T	F	T	F	F	F	T	F	F	F	F	T	F
T_3	T	T	T	T	F	T	T	F	T	F	F	T	F	F
T_4	T	F	T	T	F	F	F	F	F	T	F	T	F	T
T_5	F	T	F	T	F	T	T	F	F	F	F	F	F	F
T_6	F	F	F	T	F	F	F	T	F	F	F	F	F	F
T_7	T	T	F	T	F	F	F	T	F	T	F	T	F	F
T_8	F	T	T	F	T	F	F	F	F	F	F	F	T	F
T_9	T	T	T	T	F	F	F	T	T	F	F	T	F	T
T_{10}	T	T	T	T	F	T	T	F	F	F	T	F	F	T

Table 4: Relationship between task and fog node $R[t, N]$.

task	P_0	P_1	P_2	P_3
t_0	-1	-1	-1	-1
t_1	-1	-1	-1	-1
t_2	-1	-1	-1	-1
t_3	-1	-1	-1	-1
t_4	-1	-1	-1	-1
t_5	-1	-1	-1	-1
t_6	-1	-1	-1	-1
t_7	35.33	11.15	18.19	35.33
t_8	-1	-1	-1	-1
t_9	-1	-1	-1	-1

Table 5: Scheduled tasks *TS*.

No.	Task set
1	t_0,t_1,t_3,t_7
2	t_1,t_2,t_6,t_7
3	t_3,t_4,t_5,t_8,t_9
4	t_1,t_2,t_3,t_5,t_7,t_8
5	t_0,t_1,t_6,t_7,t_8,t_9

5. Simulation Experiment and Result Discussion

5.1. Experimental Purpose. In order to verify the TSFC algorithm proposed in this paper, we compare the performance of TSFC algorithm under the same experimental conditions with other three independent task scheduling algorithms, MCT, MET, and MIN-MIN.

5.2. Simulation Environment. Based on the simulator toolkit provided by SimGrid [41–43], the simulation environment for heterogeneous multiprocessors is built as follows:

(1) Internodes are interconnected through high speed networks.

(2) Each fog node can perform task execution at the same time and communicate with other fog nodes without competition.

(3) Every task is not preempted on the fog node.

(4) The fog nodes are heterogeneous.

The computer used in the experiment is configured as follows: Intel Core i5-3210M@2.5 GHz dual core processor, 8 GB memory. The number of the fog nodes in the experiment is 4 and 6, respectively.

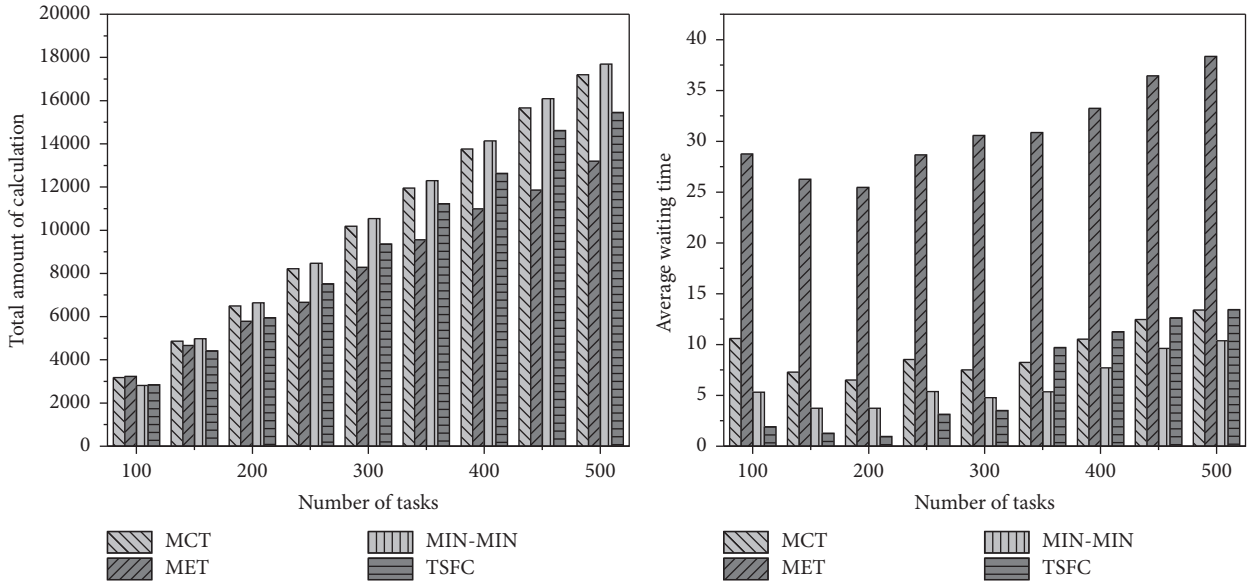
5.3. Test Data Set. The input data of TSFC algorithm include the task execution time matrix, the task scheduling relational table, and the task set. The task execution time matrix includes execution time of 10 tasks and 4 fog nodes as well as 10 tasks and 6 fog nodes. The execution time of each node is generated by a random program. The task scheduling relational table is based on the task scheduling model of fog nodes with the I-Apriori algorithm. The number of tasks in the experiment starts from 100, increasing 50 tasks each time, until the number of tasks reaches 500 tasks.

5.4. Discussion of Experimental Results

5.4.1. Result Analysis under 4 Fog Nodes. The TSFC, MCT, MET, and MIN-MIN algorithms are used to schedule the task set under 4 fog nodes, respectively. *TST* and *AWT* under different number of tasks in the four algorithms are shown in Figure 6.

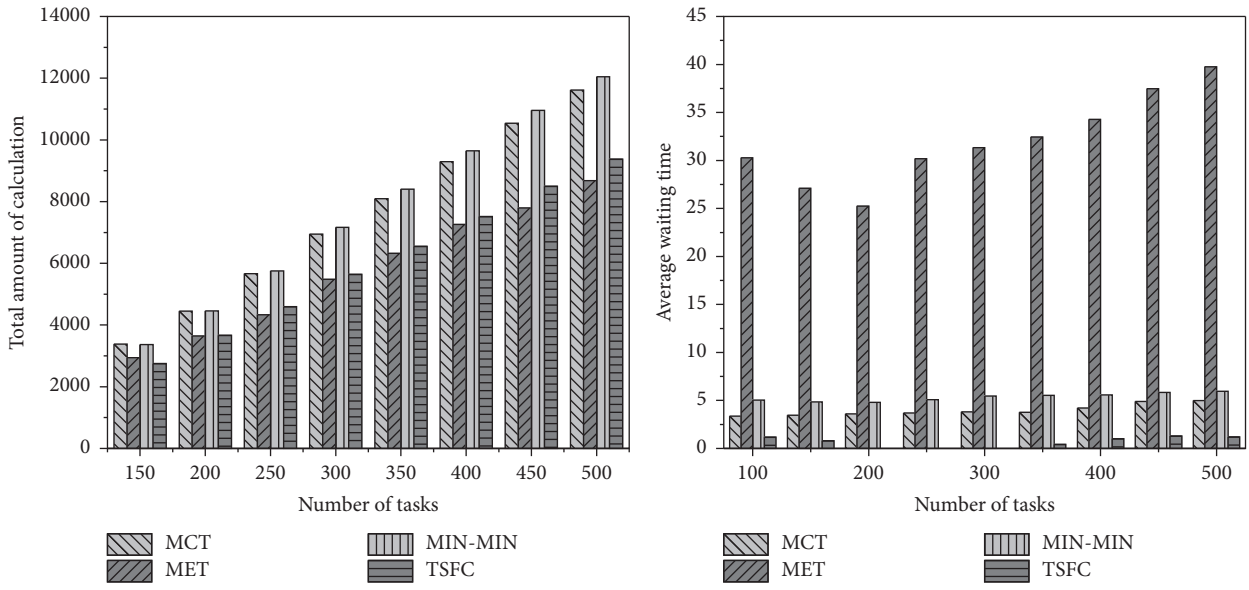
5.4.2. Result Analysis under 6 Fog Nodes. The TSFC, MCT, MET, and MIN-MIN algorithms are used to schedule the task set under 6 fog nodes, respectively. After scheduling, *TST* and *AWT* under different number of tasks in the four algorithms are shown in Figure 7.

We can see from Figures 6(a) and 7(a) that, with the number of tasks increases, the value of *TST* generated by TSFC, MCT, MET, and MIN-MIN algorithms is increasing. However, the value of *TST* generated by the TSFC algorithm is smaller than those by MCT and MIN-MIN algorithms. As the number of tasks increases, the efficiency of TSFC algorithm is higher than MCT and MIN-MIN algorithms. When the number of tasks is small, the value of *TST* generated by the TSFC algorithm is lower than that by the MET algorithm. As the number of tasks increases, the value of *TST* generated by the TSFC algorithm is larger than that by the MET algorithm. Because TSFC algorithm takes task completion time as a main parameter, as the number of tasks increases, the total completion time of scheduled tasks will be closer to the optimal solution.



(a) Comparison of execution time under 4 fog nodes (b) Comparison of average waiting time under 4 fog nodes

Figure 6: (a) Comparison of execution time under 4 fog nodes. (b) Comparison of average waiting time under 4 fog nodes.



(a) Comparison of execution time under 6 fog nodes (b) Comparison of average waiting time under 6 fog nodes

Figure 7: (a) Comparison of execution time under 6 nodes. (b) Comparison of average waiting time under 6 fog nodes.

The values of *AWT* generated by TSFC, MCT, MET, and MIN-MIN algorithms are stable from Figures 6(b) and 7(b). The value of *AWT* generated by the TSFC algorithm is less than that by MCT, MET, and MIN-MIN algorithms (the value of *AWT* of MIN-MIN algorithm in Figure 6(b) is better when the number of tasks is larger). The minimum value of *AWT* generated by TSFC algorithm in Figure 6(b) is only 3.7% of MET's, and the maximum value of *AWT* is only 35.1% of MET's. The minimum value of *AWT* generated by TSFC algorithm in Figure 7(b) is equal to 0. Because the MET algorithm takes the shortest execution time of tasks as the main scheduling parameter, the execution time of different tasks on the same fog nodes is proportional, so it will cause most of the tasks to be scheduled on the same fog node and resulting in a much higher *AWT* value. The TSFC algorithm schedules tasks which have minimum value in minimum completion time, and it shortens the value of task waiting time as much as possible, so the value of *AWT* is smaller.

In summary, the value of *TST* and *AWT* generated by TSFC algorithm is better than MCT, MET, and MIN-MIN

algorithms. The TSFC algorithm is superior to MCT, MET, and MIN-MIN algorithms in the experiments.

6. Conclusion

The fog computing is a new paradigm which attracts lots of attention. Providing satisfactory computation performance is a great challenge in the fog computing environment. In this paper, we proposed an I-Apriori algorithm by improving the Apriori algorithm. Experimental results show that the I-Apriori algorithm can improve the efficiency of generating frequent itemsets effectively. A novel task scheduling model and a novel TSFC algorithm of fog computing environment are proposed based on the I-Apriori algorithm. Association rules are generated by the I-Apriori algorithm which act as an important parameter of TSFC task scheduling algorithm. Experimental results show that TSFC algorithm has better performance than other similar algorithms in terms of task total execution time and average waiting time.

In this article, there are some other issues that do not involve, for example, bandwidth between processors, multilayer task scheduling in fog computing, and others. In future work, we will explore these areas. Furthermore, we will apply TSFC algorithm to other areas, such as oblivious RAM, string mapping, and match problems.

Data Availability

All data generated or analyzed during this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 61772205), Guangdong Province Natural Science Foundation Team Project (Grant no. 10351806001000000), Guangdong Provincial Scientific and Technological Projects (Grants nos. 2016A010101018, 2016A010119171, 2016A010106007, and 2016B090927010), Guangdong Province Advanced and Key Technology Creative Research Project (Grant no. 2014B010110004), Nansha Science and Technology Projects (Grant no. 2017GJ001), and the Industry-University-Academy Collaborative Innovation Key Project of Guangzhou (Grant no. 201604016074).

References

[1] D. Rahbari, S. Kabirzadeh, and M. Nickray, "A security aware scheduling in fog computing by hyper heuristic algorithm," in *Proceedings of the 2017 3rd Iranian Conference on Intelligent Systems and Signal Processing (ICSPIS)*, pp. 87–92, Shahrood, December 2017.

[2] C. Puliafito, E. Mingozzi, and G. Anastasi, "Fog Computing for the Internet of Mobile Things: Issues and Challenges," in *Proceedings of the 2017 IEEE International Conference on Smart Computing, (SMARTCOMP '17)*, China, May 2017.

[3] X.-Q. Pham and E.-N. Huh, "Towards task scheduling in a cloud-fog computing system," in *Proceedings of the 18th Asia-Pacific Network Operations and Management Symposium, (APNOMS '16)*, Japan, October 2016.

[4] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proceedings of the 3rd Workshop on Hot Topics in Web Systems and Technologies, (HotWeb '15)*, pp. 73–78, USA, November 2015.

[5] X. Lyu, C. Ren, W. Ni, H. Tian, and R. P. Liu, "Distributed Optimization of Collaborative Regions in Large-Scale Inhomogeneous Fog Computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 574–586, 2018.

[6] M. Huang, Y. Liu, N. Zhang et al., "A Services Routing Based Caching Scheme for Cloud Assisted CRNs," *IEEE Access*, vol. 6, pp. 15787–15805, 2018.

[7] X. Liu, M. Dong, Y. Liu, A. Liu, and N. Xiong, "Construction Low Complexity and Low Delay CDS for Big Data Code Dissemination," *Complexity*, vol. 2018, Article ID 5429546, 19 pages, 2018.

[8] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, "Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1216–1228, 2017.

[9] Q. Zhu, B. Si, F. Yang, and Y. Ma, "Task offloading decision in fog computing system," *China Communications*, vol. 14, no. 11, pp. 59–68, 2017.

[10] M. Mukherjee, L. Shu, and D. Wang, "Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges," *Communications Surveys & Tutorials*, 2018.

[11] S. Gu, Q. Zhuge, J. Yi, J. Hu, and E. H.-M. Sha, "Optimizing Task and Data Assignment on Multi-Core Systems with Multi-Port SPMs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 9, pp. 2549–2560, 2015.

[12] W. Lin, S. Xu, L. He, and J. Li, "Multi-resource scheduling and power simulation for cloud computing," *Information Sciences*, vol. 397-398, pp. 168–186, 2017.

[13] K. Chronaki, A. Rico, M. Casas et al., "Task scheduling techniques for asymmetric multi-core systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2074–2087, 2017.

[14] G. Lucarelli, F. Mendonca, and D. Trystram, "A new on-line method for scheduling independent tasks," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGRID '17)*, pp. 140–149, Spain, May 2017.

[15] J. Wu and X.-J. Hong, "Energy-Efficient Task Scheduling and Synchronization for Multicore Real-Time Systems," in *Proceedings of the IEEE 3rd international conference on big data security on cloud*, pp. 179–184, China, May 2017.

[16] C. Tang, X. Wei, S. Xiao et al., "A Mobile Cloud Based Scheduling Strategy for Industrial Internet of Things," *IEEE Access*, vol. 6, pp. 7262–7275, 2018.

[17] T. Li, Y. Liu, L. Gao, and A. Liu, "A cooperative-based model for smart-sensing tasks in fog computing," *IEEE Access*, vol. 5, pp. 21296–21311, 2017.

[18] Y. Liu, J. E. Fieldsend, and G. Min, "A Framework of Fog Computing: Architecture, Challenges, and Optimization," *IEEE Access*, vol. 5, pp. 25445–25454, 2017.

[19] H. Wang, W. Wang, Z. Cui, X. Zhou, J. Zhao, and Y. Li, "A new dynamic firefly algorithm for demand estimation of water resources," *Information Sciences*, vol. 438, pp. 95–106, 2018.

[20] W. Lin, S. Xu, J. Li, L. Xu, and Z. Peng, "Design and theoretical analysis of virtual machine placement algorithm based on peak workload characteristics," *Soft Computing*, vol. 21, no. 5, pp. 1301–1314, 2017.

[21] W. Chen, L. Peng, J. Wang et al., "Inapproximability results for the minimum integral solution problem with preprocessing over infinity norm," *Theoretical Computer Science*, vol. 478, pp. 127–131, 2013.

[22] Y. Wang, K. Li, and K. Li, "Partition Scheduling on Heterogeneous Multicore Processors for Multi-dimensional Loops Applications," *International Journal of Parallel Programming*, vol. 45, no. 4, pp. 827–852, 2017.

[23] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, "Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF)," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 64, no. 1, pp. 280–285, 2015.

[24] R. Bruni and G. Bianchi, "Effective Classification Using a Small raining Set Based on iscretization and Statistical Analysis," *IEEE Transactions On knowledge and data engineering*, vol. 27, no. 9, pp. 2349–2361, 2015.

[25] C. Zhou, B. Cule, and B. Goethals, "Pattern Based Sequence Classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1285–1298, 2016.

[26] B. Tang, H. He, P. M. Baggenstoss, and S. Kay, "A Bayesian Classification Approach Using Class-Specific Features for Text Categorization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1602–1606, 2016.

[27] Z. Liu, Y. Huang, J. Li, X. Cheng, and C. Shen, "DivORAM: Towards a practical oblivious RAM with variable block size," *Information Sciences*, vol. 447, pp. 1–11, 2018.

[28] B. Li, Y. Huang, Z. Liu, J. Li, Z. Tian, and S. Yiu, "HybridORAM: Practical oblivious cloud storage with constant bandwidth," *Information Sciences*, 2018.

[29] W. Chen, Z. Chen, N. F. Samatova, L. Peng, J. Wang, and M. Tang, "Solving the maximum duo-preservation string mapping problem with linear programming," *Theoretical Computer Science*, vol. 530, pp. 1–11, 2014.

[30] Y. Huang, W. Li, Z. Liang, Y. Xue, and X. Wang, "Efficient business process consolidation: combining topic features with structure matching," *Soft Computing*, vol. 22, no. 2, pp. 645–657, 2018.

[31] W. Lin, C. Zhu, J. Li, B. Liu, and H. Lian, "Novel algorithms and equivalence optimisation for resource allocation in cloud computing," *International Journal of Web and Grid Services*, vol. 11, no. 2, pp. 69–78, 2015.

[32] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999.

[33] T. D. Brauny, H. Siegely, N. Becky et al., "A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems," *parallel & distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[34] Y. Li, G. Wang, L. Nie, Q. Wang, and W. Tan, "Distance metric optimization driven convolutional neural network for age invariant face recognition," *Pattern Recognition*, vol. 75, pp. 51–62, 2018.

[35] M. Xiao, Y. Yin, Y. Zhou, and S. Pan, "Research on improvement of apriori algorithm based on marked transaction compression," in *Proceedings of the 2nd IEEE Advanced Information Technology, Electronic and Automation Control Conference, (IAEAC '17)*, pp. 1067–1071, China, March 2017.

[36] V. S. Tseng, C.-W. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining the concise and lossless representation of high utility itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 726–739, 2015.

[37] W. Lin, Z. Wu, L. Lin, A. Wen, and J. Li, "An ensemble random forest algorithm for insurance big data analysis," *IEEE Access*, vol. 5, pp. 16568–16575, 2017.

[38] J. Yang, H. Huang, and X. Jin, "Mining web access sequence with improved apriori algorithm," in *Proceedings of the 20th IEEE International Conference on Computational Science and Engineering and 15th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, CSE and EUC 2017*, pp. 780–784, China, July 2017.

[39] S. Zhang, Z. Du, and J. T. L. Wang, "New techniques for mining frequent patterns in unordered trees," *IEEE Transactions on Cybernetics*, vol. 45, no. 6, pp. 1113–1125, 2015.

[40] D. Hoang and T. D. Dang, "FBRC:Optimization of task scheduling in Fog-based Region and Cloud," in *Proceedings of the 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 11th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Conference on Embedded Software and Systems, Trustcom/BigDataSE/ICESS 2017*, pp. 1109–1114, Australia, August 2017.

[41] C. A. Brennand, J. M. Duarte, and A. P. Silva, "SimGrid: A simulator of network monitoring topologies for peer-to-peer based computational grids," in *Proceedings of the 2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1–6, Medellin, Colombia, November 2016.

[42] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI Applications: The SMPI Approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2387–2400, 2017.

[43] A. Mohammed, A. Eleliemy, and F. M. Ciorba, "Towards the Reproduction of Selected Dynamic Loop Scheduling Experiments Using SimGrid-SimDag," in *Proceedings of the 19th international conference on high performance computing and communications; IEEE 15th international conference on smart city; IEEE 3rd international conference on data science and systems*, pp. 623–626, Bangkok, December 2017.