Future Generation Computer Systems 113 (2020) 474–487

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

PFPMine: A parallel approach for discovering interacting data entities in data-intensive cloud workflows

Yuze Huang[a], Jiwei Huang[b,c,*], Cong Liu[d], Chengning Zhang[e]

[a] School of Information Science and Engineering, Chongqing Jiaotong University, Chongqing 400074, China
[b] Department of Computer Science and Technology, China University of Petroleum-Beijing, Beijing 102249, China
[c] Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum-Beijing, Beijing 102249, China
[d] College of Computer Science and Technology, Shandong University of Technology, Zibo 255300, China
[e] Grab Company, Singapore 573972, Singapore

ARTICLE INFO

Article history:
Received 25 December 2019
Received in revised form 2 June 2020
Accepted 7 July 2020
Available online 11 July 2020

Keywords:
Data entity discovery
MapReduce
Data-intensive workflow
Cloud computing

ABSTRACT

With the evolution of cloud computing, communities and companies deployed their workflows on cloud to support end-to-end business processes that are usually syndicated with other external services. To improve the efficiency of the system as well as reducing energy consumption, data placement and backup strategies should be carefully designed. One of the most challenging problems is the discovery of interacting data entities in date-intensive workflows. To tackle this challenge, this paper presents a frequent pattern-based approach named FPMine for interacting data entity discovery in cloud workflows. A direct discriminative mining algorithm is first proposed to determine the minimum support threshold, based on which FP-tree is constructed to formulate the frequent item pairs. Next, FP-matrix is applied to avoid traversing the FP-trees during data entity discovery, and a pruning approach is introduced to reduce the redundancy of frequent item pairs. Furthermore, we propose a parallel data entity mining algorithm using MapReduce framework, namely PFPMine, and then design a primitive data placement and backup strategy. Finally, we evaluate the efficiency of our approach by experiments using real-life data, based on which we show that our approach can facilitate the discovery of interacting data entities with efficiency for cloud workflows. Comparing with traditional FP-growth approach, we pay only a multiplicative factor for making our approach able to extract fine-grained frequent item pairs rather than frequent patterns, which can bring significant advantages to data placement. After parallelization, the PFPMine algorithm performs better with high efficiency for both sparse datasets and dense datasets than FP-growth. The results show that PFPMine can reduce the running time by at least 25%, and preforms with significantly higher efficiency than FP-growth approach.

1. Introduction

With the rapid development of the Internet, large volumes of data, both structured and unstructured, are processed, analyzed, and stored for various applications. Data-intensive systems are widely used to deal with large amount of data to support the rapidly changed complex business goal [1]. They have been applied in different applications, such as sensor-based data collection and analytics [2], intelligent transportation systems [3], and cyber-physical systems [4] *etc.*. Applications in data-intensive system can be formulated by a serial of business processes, each of which accomplishes one piece of sub-task and composes to a complete workflow. In such cases, both procedures and data should be jointly considered.

At the very beginning, most traditional research on workflow modeling and implementation focused on control flow instead of data flow issues, lacking the ability to formulate the operations (read, write, add, remove, *etc.*) of global data during the procedures of the design and implementation for systems. With the emergence of data-intensive applications and systems, novel data-aware (also called data-centric) business process models have been proposed to fill this gap [5]. A notable exponent named artifact-centric model was produced by IBM researchers [6]. In such model, data entity (also called business entity or artifact) records the core information for business operations. It is well-known that the data entities which are manipulated by services are not independent, since they always interact with each other during the execution of the business process.

* Corresponding author.
E-mail addresses: huangyz@cqjtu.edu.cn (Y. Huang), huangjw@cup.edu.cn (J. Huang), liucongchina@sdust.edu.cn (C. Liu).

https://doi.org/10.1016/j.future.2020.07.018
0167-739X/© 2020 Elsevier B.V. All rights reserved.

Cloud computing is able to deliver computing as well as storage resources as a kind of public facility [7], and it brings significant efficiency to workflows due to the high utilization of the resources [8]. With the development of cloud computing, more and more communities have deployed their workflows into the cloud environment. However, for workflows that provide support for data-aware business processes, some new challenges have caught wide concern.

Data placement is one of the most urgent issues that need to be solved in cloud workflows [9]. Many applications that collaborate with different organizations or services have to deal with data that is geographically dispersed in different data centers [10–13]. Data manipulations that involve different data centers require a large amount of data communication which is quite resource-consuming and time-consuming. Hence, after the development of workflow, we should design data placement strategy to move the data items closer to the geo-distributed organizations while reducing the communication overhead [14]. Another challenge in cloud workflows is data backup. Since the requirements on reliability and security of both services and data are growing rapidly in cloud applications, backup data brings more and more workload to data centers [15]. It has been reported by Facebook that there have been nearly one trillion posts available on its websites involving a significantly large amount of data storage in the underlying data centers.[1] Recently, another official report posted by Alibaba which is one of the biggest online merchants in China showed incredible numbers in e-commerce.[2] It has been recorded that a total of 7.4 billion mobile interactions were processed by their systems at the peak rate of 175,000 orders per second during the 24-hour online sale on Nov. 11, hitting a new record as $17.79 billion in gross merchandise volume (GMV). All of these show the impossibility for backuping all the data in cloud-based workflow without distinction. Furthermore, data entities in business processes may be with different importance levels, and thus the data entity backup strategy should be carefully designed to achieve the cost-effectiveness.

To solve these problems, the data in business processes should be considered differently according to its importance and involved activities. Therefore, as a foundation for managing data in cloud workflow, interacting data entities in business processes should be precisely discovered to provide theoretical support for both data placement and backup. Due to the discovered results obtained by our approach, interacting data entities are placed into a data center based on the capacity constraint of data center, and thus the communication overhead is reduced accordingly. Although several researches have been dedicated to solving data problems in reality, few of them focused on data entity discovery, and thus the fine-grained frequent item pairs (also called frequent 2-itemsets) of the data affiliated to business processes cannot be precisely extracted. We also noticed that the traditional FP-growth algorithm [16] cannot discover the frequent 2-itemsets directly, although some Apriori-like algorithms that can obtain the frequent item pairs by join arithmetic, large number of redundancy itemsets can be generated during the discovery procedure, which may cost large amount of time and resources [17]. Therefore, an algorithm for discovering the frequent 2-itemsets in cloud workflow directly should be carefully designed.

In this paper, a frequent pattern based approach is presented for discovering interacting data entities in cloud workflows. In our approach, an algorithm is produced to discover the discriminative frequent 2-itemsets directly. This algorithm determines the minimum support threshold at first, then, an additional structure is constructed to record the frequency information of the dataset, based on which the algorithm discovers the discriminative itemsets with interestingness measure technique. Finally, we parallelize the algorithm using MapReduce framework, with which the efficiency of our approach is improved significantly. Specifically, the contributions of this paper can be described as the following threefold.

- A frequent pattern based approach is proposed to discover interacting data entities. In this approach, an algorithm named FPMine is proposed, which determines the minimum support threshold automatically at first, and then FP-tree and FP-matrix are constructed to improve the efficiency of our algorithm.
- A pruning algorithm with interestingness measure is produced to search the discriminative frequent 2-itemsets, which cannot generate the redundancy frequent itemsets.
- A parallel algorithm, called PFPMine, is proposed. This approach parallelizes the FPMine algorithm using MapReduce framework, based on which the efficiency of our approach is improved significantly.

The rest of this paper is organized as follows. Section 2 introduces the basic definitions and background information. Section 3 presents an overall framework for the data entity discovery process, and then a frequent pattern based algorithm is produced in this section. Section 4 proposes the frequent pattern based parallel mining algorithm using MapReduce framework to improve the performance. A primitive data placement and backup strategy is presented to investigate the benefits of our approach in Section 5. Furthermore, the computational complexity of our algorithms and the experimental results are shown in Section 6. Finally, the related work of this research is introduced in Section 7, and then we conclude this paper in Section 8.

2. Preliminaries

In this section, we present the fundamental concepts and definitions of this paper, which are the notion of workflow event log and the definition of frequent itemsets in business process.

The input of our approach is the workflow event log, which records the execution status of the workflow. Table 1 gives an example of the workflow event log of an online shopping process. The workflow event log indicates the specific information of the workflow execution status. We noticed that a workflow event log consists of numerous cases (also called transactions), and each case consists of numerous services (also called events) such that each service relates to precisely one case. Services within a case are ordered [18]. Formally, we define the workflow event log as Definition 1.

Definition 1 (*Workflow Event Log*)**.** A workflow event log *EL* is defined by a 5-tuple $C \triangleq (Cid, Sid, TS, \prod, \sum)$ where,

- *Cid* is the nonempty set of cases ID.
- *Sid* is the nonempty set of services ID.
- *TS* is the finite serial of time-stamps, which is used to sort the services in the logs.
- $\prod$ is the finite serial of services, in which each service corresponds to a service ID.
- $\sum$ is the finite serial of data attributes, which involved by the services.

Table 2 shows the compact representation of a workflow log, where each case is represented by a sequence of data attributes. For brevity, we transformed the data attributes into single-letter

[1] https://techcrunch.com/2014/12/28/mining-the-hive-mind/.

[2] http://www.alizila.com/infographic-double-11-sales-and-engagement-soar/.

Table 1
An example of workflow event log.

Cid	Sid	TimeStamp	Service	Data attributes
001	S001	10-12-2018,15:05	Picking Products	Customer=John; ProID=12100015; Product=iPhone8; NumofItems=1; Price=4688
	S002	10-12-2018,15:07	Selecting Coupons	CouponType=Discount; CouponCount=300; Price=4388
	S003	10-12-2018,15:08	Shipping	ShipType=EMS ShipCount=0
	S004	10-12-2018,15:09	Placing Orders	Customer=John OrderID=20181210086 Address=CQJTU Contact=010-98862256
	S005	10-12-2018,15:10	Payment	Customer=John; Price=4388; PayID=2018121000123
	......	......	......	......

Table 2
A compact form of workflow event log.

Case ID	Data Attributes
001	$< a, b, c, e, f, g, h, i, j, k, l, m >$
002	$< a, b, c, d, e, h, i >$
003	$< a, b, c, d, e, f, g >$
004	$< a, b, c, d, h, i >$
......	......

labels (*i.e.*, “a” represents the “Customer”, “b” represents the “ProID”, *etc.*).

The objective of our approach is to find potential frequent itemsets from the data attributes. Here, the definition of frequent itemset in a business process can be found as follows.

Definition 2 (*Frequent Itemset in Business Process*)**.** Let $I = \{i_1, i_2, i_3, \ldots\ldots, i_m\}$ be a set of data attributes in business process. Let $T = \{T_1, T_2, T_3, \ldots\ldots, T_k\}$ be a nonempty set of business process cases (also called transactions), where each case T_i is associated with an ID, represented by an element in *Cid*. Assuming the original dataset D contains a set of n instance $D = \{Cid_i, T_i\}_{i=1}^{n}$, where each case contains a set of items $X \subseteq I$. An itemsets $\alpha = \{\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_l\}$ is a subset of I. We say $\{(\alpha_i, \alpha_j)\}$ is frequent itemset in business process if and only if existing $sup(\alpha_i) \geqslant min\_sup$, $sup(\alpha_j) \geqslant min\_sup$, and the interestingness measure value of $\{(\alpha_i, \alpha_j)\}$ is larger than the threshold ξ.

Data entity records the key information for business operations in workflow. After discovering the data entities, we need to model these data entities and place them into the data centers. The model of data entity can be formalized as the descriptions of our previous work [19].

3. Frequent pattern based data entities discovery

In previous section, we give the definitions and notations of workflow event log and frequent itemsets in business process. Next, we produce a direct discriminative mining algorithm to discover the frequent 2-itemsets in business process.

3.1. Framework of discovery approach

In order to reveal the potential relationship between data entities, we propose a frequent pattern-based approach, namely FPMine, which can be described as the following steps, illustrated by Fig. 1.

Step 1 Firstly, data attributes are extract from workflow event log as the original data set of our algorithm.
Step 2 A minimum support threshold is calculated by analyzing the original data set. Data items with support count above the threshold are extracted to construct an FP-tree with FP-matrix. Details can be found in Sections 3.2 and 3.3.

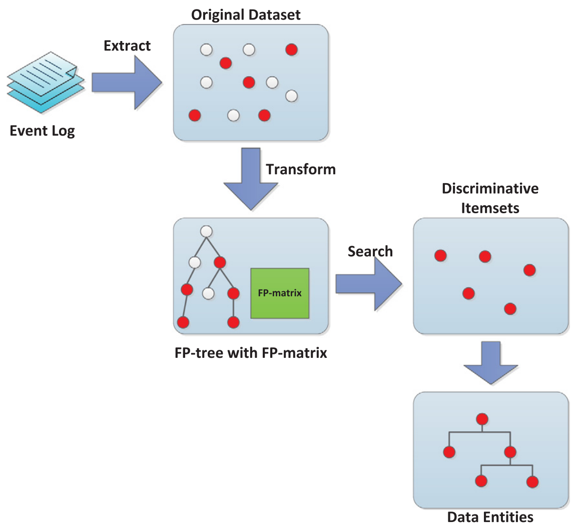


Fig. 1. The framework of FPMine.

Step 3 Based on the FP-tree construction, a pruning algorithm with interestingness measure is produced to searching the discriminative frequent 2-itemsets.
Step 4 Interacting data entities are modeled according to the descriptions of our previous work [19], and then the frequent item sets are stored into the data center according to the primitive strategy produced in Section 5.

3.2. Calculation of support threshold

The foundation of frequent pattern discovery is how to calculate a precise minimum support threshold. However, many existing approaches for frequent pattern discovering left the task of threshold calculation for users [16,20]. Although there have been some efforts for automatically finding a threshold, most of them are based on exhaustive search or supervised machine learning resulting in their inefficiency [21,22]. In this paper, we present an automatic threshold calculation approach according to the distribution character of data items without training data to fill this gap.

According the statistic character of data attributes in workflow event log, we suggest that the frequency of the data is followed by the Zipf-like distribution [23], which can be regarded as a discrete counter part of the Pareto distribution [24]. The Cumulative Distribution Function (CDF) of Zipf distribution can be expressed as

$$F(k, N, s) = \frac{\sum_{n=1}^{k} \frac{1}{n^s}}{\sum_{n=1}^{N} \frac{1}{n^s}} \tag{1}$$

where $k \in \{1, 2, \ldots, N\}$ is the frequency rank, $N \in \{1, 2, \ldots\}$ is the number of elements, and $s \geq 1$ is the value of the exponent characterizing the distribution. In light of the characters of Zipf distribution, we assume p is the proportion of the minimum support threshold. When s is fixed, p can be obtained by solving the following equation.

$$F(pN, N, s) = \frac{\sum_{n=1}^{pN} \frac{1}{n^s}}{\sum_{n=1}^{N} \frac{1}{n^s}} = 1 - p \tag{2}$$

The procedures for determining the minimum support threshold are illuminated in Algorithm 1. Firstly, the workflow log is scanned to obtain the frequency of each data attribute. Secondly, we rank the items in a descending order. Finally, p can be calculated according to Eq. (2), and then $min\_sup = F_{\lfloor pN \rfloor}$, where min_sup represents the minimum support threshold, and $F_{\lfloor pN \rfloor}$ represents the frequency of item $\lfloor pN \rfloor$.

Algorithm 1 Algorithm for Determining Minimum Support Threshold

```
Input: Workflow Log $EL$
Output: Minimum Support Threshold $min\_sup$
1: Scan the workflow log to get the frequency of each data attribute
2: Rank the items in descending order
3: Calculate $p$ by Eq. (2). and then the minimum support threshold can be obtained according to $min\_sup = F_{\lfloor pN \rfloor}$
```

3.3. FP-tree with FP-matrix construction

FP-growth [16] is a classical algorithm for mining frequent itemsets. Its basic idea is to traverse FP-tree which is a well-defined tree-form data structure recording data frequency information in a depth-first order. However, it has been reported that nearly 80% of the computational resources are consumed for traversing FP-tree in FP-growth [20]. To improve its efficiency, we design a novel data structure, namely FP-matrix, to store the frequency information and interestingness measure value of item pairs. Furthermore, we redesign the algorithm for finding frequent item pairs from workflow logs, whose performance can be significantly improved with FP-matrix. In the following, we start with introducing the basic definition of the FP-matrix.

Definition 3 (*FP-Matrix*)**.** Let $X = \{x_1, x_2, x_3, \ldots\ldots, x_m\}$ be a set of data items with support count above the support threshold. An FP-matrix M is a $(m-1) \times (m-1)$ matrix, where each element of the matrix can be defined by a tuple $M_{[x_i,x_j]} = (C_{x_i,x_j}, V_{x_i,x_j})$, where

- C_{x_i,x_j} is the support count of an ordered pair $\{(x_i, x_j)\}$ in X.
- V_{x_i,x_j} is the interestingness measure value of an ordered pair $\{(x_i, x_j)\}$ in X.

In order to efficiently store the support counts and interestingness values, M is defined as a triangular matrix, where its first index represents items x_1 to x_{m-1} while the second index represents items x_2 to x_m.

For clarity, we explain the construction of the FP-matrix by an example. Fig. 2(a) shows a dataset which is extracted from the workflow event log. At the beginning, each element of FP-matrix is defined as a symbol Φ. After the first traverse of the original dataset, we sort the data items as $a:5, b:4, d:4, g:3, c:2, e:2$, which is shown in the header table of Fig. 2(b). Then, we construct the FP-tree and FP-matrix at the same time during the second traverse of the dataset.

The FP-tree and FP-matrix can be found in Fig. 2(b) and 2(c). During the second traverse for constructing the FP-tree with FP-matrix, all data items in the cases are extracted from the dataset, and then the data items are sorted according to the order shown in header table of Fig. 2(b). In this algorithm, we scan the cases in the dataset in turn, and insert the items into the FP-tree P, at the same time, C_{x_i,x_j} is incremented by 1 if $\{(x_i, x_j)\}$ is contained in this case. For instance, for the third case, we extract the items $\{a, b, c, d\}$ from this case, the items are inserted into the FP-tree according to the FP-growth algorithm for constructing the FP-tree [16], at the same time, the pairs in this case are inserted into the FP-matrix, the corresponding elements of the FP-matrix are all incremented by 1. After the traverse of all the cases of the dataset. The FP-matrix contains all frequency information of the dataset.

Algorithm 2 Algorithm for Constructing FP-tree with FP-matrix

```
Input: Original dataset $D$; Minimum support threshold $min\_sup$
Output: FP-tree $P$; FP-matrix $M$
1: Define the list of frequent items:$L \leftarrow \Phi$
2: for all cases $T_i \subseteq D$ do
3:   if item $a.count \geq min\_sup$ then
4:     Insert $a$ into $L$
5:   end if
6: end for
7: Sort items of $L$ in descending order by support count.
8: Create the root of the FP-tree $P$, and label it as "Root"
9: Create the FP-matrix $M$, and let $C_{x_i,x_j} \leftarrow \Phi$, $V_{x_i,x_j} \leftarrow \Phi$
10: for all cases $T_i \subseteq D$ do
11:   Sort the frequent items in $T$ in descending order by support count
12:   INSERT_TREE($T_i$, $P$)
13:   INSERT_MATRIX($T_i$, $M$)
14: end for
```

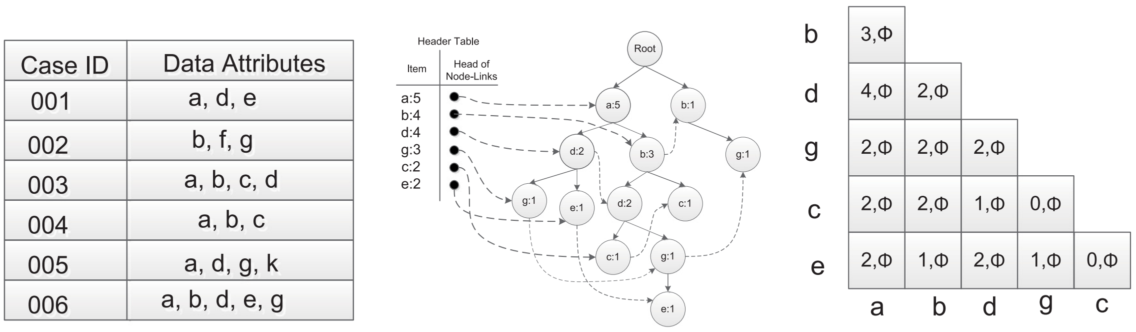
Algorithm 3 Procedure for INSERT_TREE(T_i, P) [16]

```
INSERT_TREE($T_i$, $P$)
1: Let the sorted data item list in $T_i$ be $[e|E]$, where $e$ is the first element and $E$ is the remaining list, $N.count \leftarrow 0$
2: if FP-tree $P$ has a child $N$, and $N.item\_name = e.item\_name$ then
3:   $N.count \leftarrow N.count + 1$
4: else
5:   Create a new node $N$, and $N.count \leftarrow 1$
6:   $N.parent \leftarrow P$
7:   let $N$ link to the node with the same $item\_name$ via the node-link structure
8: end if
9: if $E \neq \emptyset$ then
10:   INSERT_TREE($E$, $N$)
11: end if
```

Algorithm 4 Procedure for INSERT_MATRIX(T_i, M)

```
INSERT_MATRIX($T_i$, $M$)
1: $C_{x_i,x_j} \leftarrow \Phi$
2: for $i \leftarrow 1$ to $k$ do
3:   for $j \leftarrow i + 1$ to $k$ do
4:     if $\{x_i, x_j\} \in T_i$ then
5:       $C_{x_i,x_j} \leftarrow C_{x_i,x_j} + 1$
6:     end if
7:   end for
8: end for
```

The procedures for constructing the FP-tree with FP-matrix are shown in Algorithm 2. The key functions INSERT_TREE(T_i, P) and INSERT_MATRIX(T_i, M) are illuminated in Algorithms 3 and 4.



Case ID	Data Attributes
001	a, d, e
002	b, f, g
003	a, b, c, d
004	a, b, c
005	a, d, g, k
006	a, b, d, e, g

$$
\begin{array}{c|ccccc}
b & 3,\Phi & & & & \\
d & 4,\Phi & 2,\Phi & & & \\
g & 2,\Phi & 2,\Phi & 2,\Phi & & \\
c & 2,\Phi & 2,\Phi & 1,\Phi & 0,\Phi & \\
e & 2,\Phi & 1,\Phi & 2,\Phi & 1,\Phi & 0,\Phi \\
\hline
 & a & b & d & g & c
\end{array}
$$

(a) Dataset (b) FP-tree of Dataset (c) FP-matrix of Dataset

Fig. 2. An example of constructing FP-tree with FP-matrix.

After constructing the FP-tree and FP-matrix, in the procedure of mining discriminative frequent itemsets, there is no need to traverse the dataset to obtain the frequency information of the itemset. We only need to retrieve the FP-matrix, it can reduce the computational complexity of our algorithm effectively.

Next, we analyze the space complexity of the algorithm for constructing FP-tree and FP-matrix. Assuming m is the quantity of frequent items in the header table. Traditional matrix is an $m \times m$ matrix, but there are some repetitive elements in the matrix, which may lead to high complexity. The FP-matrix is constructed as a triangular matrix to reduce the storage space. The FP-matrix is a $(m-1) \times (m-1)$ echelon matrix, whose size is $\sum_{i=1}^{m-1} i = \frac{m(m-1)}{2}$. The storage space can be reduced more than 50%. It is obvious that the algorithm can reduce the time complexity without increasing the storage space significantly at the same time.

In summary, we construct the FP-matrix at the phase of constructing the FP-tree. The FP-matrix stores the frequent counts and interestingness measure values of all item pairs. Based on the FP-matrix construction, we only need to retrieve the FP-matrix instead of traversing the FP-tree during the procedure for discovering frequent 2-itemsets, which can reduce the time complexity without increasing the storage space significantly at the same time.

3.4. Direct discriminative mining algorithm

With the algorithm for constructing the FP-tree with FP-matrix presented, we directly mine the discriminative itemsets in this sub-section. Traditional frequent pattern mining algorithms often adopt a two-step approach, where it first generates the complete frequent itemset, and then selects the discriminative patterns from such candidate itemset. Unlike the traditional frequent pattern mining algorithm, our algorithm mainly focus on mining the frequent item pairs other than frequent patterns. Therefore, a new algorithm should be designed accordingly.

Most of existing frequent pattern mining algorithms adopt a support-confidence framework. They used a threshold to lower bound the support count or confidence value of the frequent patterns. Although they worked in many of the cases in reality, a bad threshold may lead to numerous unexpected non-interesting results, resulting in critical drawback to the effectiveness as well as the performance of the algorithms. To fill this gap, several interestingness measures have been proposed, *e.g.*, χ^2, *Lift*, *Cosine*, and *Kulczynski etc.* [25]. In our algorithm, we adopt the *Kulczynski* measure (also denote as *Kulc* for simplicity) to evaluate the correlations among data items, whose definition can be found as follows.

Definition 4 (*Kulczynski Measure*)**.** *Kulczynski* is a null-invariant measure. Given two items, x and y, the *Kulczynski* measure is defined as.

$$Kulc(x, y) = \frac{1}{2}(P(x|y) + P(y|x)) \tag{3}$$

Kulc can be regarded as the average of two confidence measures values. With the definitions introduced in this paper, *Kulc* measure can be expressed in terms of support count, as shown in Theorem 1.

Theorem 1. *Given two items, x and y, the* Kulc *measure of x and y is represented by the support count.*

$$Kulc(x, y) = \frac{sup(xy)}{2}\left(\frac{1}{sup(x)} + \frac{1}{sup(y)}\right) \tag{4}$$

where $sup(xy)$ means the support count of the pair of $\{(x, y)\}$, which can be obtained from FP-matrix. The $sup(x)$ and $sup(y)$ represent the support count of the items x and y, respectively.

Proof. Eq. (3) can be expanded as the following form.

$$
\begin{aligned}
Kulc(x, y) &= \frac{1}{2}\left(\frac{P(xy)}{P(y)} + \frac{P(xy)}{P(x)}\right) \\
&= \frac{1}{2}\left(\frac{\frac{sup(xy)}{n}}{\frac{sup(y)}{n}} + \frac{\frac{sup(xy)}{n}}{\frac{sup(x)}{n}}\right) \\
&= \frac{sup(xy)}{2}\left(\frac{1}{sup(x)} + \frac{1}{sup(y)}\right) \quad \square
\end{aligned} \tag{5}
$$

In our algorithm, we select the positively correlated items with the *Kulc* measure. Given two items x and y, suppose that items x and y are both frequent, that is, $sup(x) \geq min\_sup$ and $sup(y) \geq min\_sup$, where min_sup is the minimum support threshold. If the *Kulc* measure of x and y, $Kulc(x, y) > \xi$, we say the items x, y are positively correlated. If $Kulc(x, y) < \xi$, we believe the items x and y are negatively correlated. If the $Kulc(x, y) = \xi$, we believe the items of x and y are independent. Here ξ is the negative pattern threshold that can be obtained from users or by some automatic techniques [25].

Based on the former definitions and theorems, we design a branch-and-bound algorithm for directly mining discriminative frequent 2-itemsets. In this algorithm, a queue which stores frequent items with an increasing order according to their support count is created, and the positively correlated itemsets are searched by calculating the interestingness measure value. During this procedure, the *Kulc* value for each pair of the items in the branch is calculated. If the *Kulc* value is larger than the negative pattern threshold ξ, then our algorithm selects the item pair out, otherwise skipping them, and so fourth. Algorithm 5 shows the detail procedures of mining discriminative itemsets, whose key function BRANCH_SEARCH(x, y, z, M) is illuminated in Algorithm 6.

Algorithm 5 Algorithm for Mining Discriminative Itemsets

```
Input: A non-empty FP-tree $P$; FP-matrix $M$
Output: Discriminative itemsets $\beta$
1: Construct a queue $Q$, and sort items as support increasing order
2: $\beta \leftarrow \emptyset$
3: while $x \leftarrow DeQueue(Q)$ do
4:   Let $y \leftarrow x.parent, z \leftarrow y.parent$
5:   while $x \neq null$ do
6:     while $y \neq Root$ do
7:       if $V_{x,y} = \Phi$ then
8:         $V_{x,y} \leftarrow Kulc(x, y)$
9:       end if
10:      $\beta \leftarrow \beta \cup$ BRANCH_SEARCH$(x, y, z, M)$
11:    end while
12:    $x \leftarrow x.Link$
13:  end while
14: end while
```

Algorithm 6 Procedure for BRANCH_SEARCH(x, y, z, M)

```
BRANCH_SEARCH$(x, y, z, M)$
1: if $z = Root$ then
2:   if $V_{x,y} > \xi$ then
3:     return $\{(x, y)\}$
4:   end if
5: end if
6: $\beta \leftarrow \emptyset$
7: if $sup(xy) = sup(xz)$ then
8:   if $sup(y) = sup(z)$ then
9:     if $V_{x,y} > \xi$ then
10:      $\beta \leftarrow \beta \cup \{(x, y)\} \cup \{(x, z)\}$
11:    end if
12:    Let $y \leftarrow z.parent, z \leftarrow y.parent$
13:  else
14:    if $V_{x,y} \leqslant \xi$ then
15:      Let $y \leftarrow z.parent, z \leftarrow y.parent$
16:    else
17:      $\beta \leftarrow \beta \cup \{(x, y)\}$
18:      Let $y \leftarrow y.parent, z \leftarrow y.parent$
19:    end if
20:  end if
21: end if
22: if $sup(xy) < sup(xz)$ then
23:   if $V_{x,y} > \xi$ then
24:     $\beta \leftarrow \beta \cup \{(x, y)\} \cup \{(x, z)\}$
25:     Let $y \leftarrow z.parent, z \leftarrow y.parent$
26:   else
27:     Let $y \leftarrow y.parent, z \leftarrow y.parent$
28:   end if
29: end if
30: if $sup(xy) > sup(xz)$ then
31:   if $V_{x,y} > \xi$ then
32:     $\beta \leftarrow \beta \cup \{(x, y)\}$
33:     Let $y \leftarrow y.parent, z \leftarrow y.parent$
34:   else
35:     Let $y \leftarrow z.parent, z \leftarrow y.parent$
36:   end if
37: end if
38: return $\beta$
```

We assume that there is a branch γ in the FP-tree, in this branch, there exists a path from bottom to up, that is x, y, z. Obviously, the support count of these three items has the following relationship

$$sup(x) \leq sup(y) \leq sup(z) \tag{6}$$

From the Eq. (4), we know the *Kulc* measures of item pairs {(x, y)} and {(x, z)} can be represented as

$$Kulc(x, y) = \frac{sup(xy)}{2}\left(\frac{1}{sup(x)} + \frac{1}{sup(y)}\right) \tag{7a}$$

$$Kulc(x, z) = \frac{sup(xz)}{2}\left(\frac{1}{sup(x)} + \frac{1}{sup(z)}\right) \tag{7b}$$

Following, we will analyze all kinds of conditions in our algorithm. According to the relationship of $sup(xy)$ and $sup(xz)$, it can be categorized to three conditions as follows.

(1) $sup(xy) = sup(xz)$

In this condition, we analyze the relationship of $sup(y)$ and $sup(z)$. If $sup(y) = sup(z)$, then $Kulc(x, y) = Kulc(x, z)$. Therefore, our task can be transferred to the estimation of the correlations between items x and y.

If $sup(y) < sup(z)$, then $Kulc(x, y) > Kulc(x, z)$. In this condition, if $Kulc(x, y) \leqslant \xi$, so $Kulc(x, z) \leqslant \xi$. Then we only need calculate $Kulc(y, z)$ and compare the relationship between $sup(x)$ and $sup(y)$. Otherwise we should calculate $Kulc(x, z)$ to estimate whether the item pair $\{(x, z)\}$ is positively correlated.

(2) $sup(xy) < sup(xz)$

In this condition, no matter the relationship between $sup(y)$ and $sup(z)$. It is obvious that $sup(xy) \leq sup(y)$, so we can deduce that $Kulc(x, y) < Kulc(x, z)$. If $Kulc(x, y) > \xi$, the item pair $\{(x, y)\}$ is positively corrected, we can deduce that $\{(x, z)\}$ is also positively correlated. Otherwise we must calculate $Kulc(x, z)$ to estimate whether the item pair $\{(x, z)\}$ is positively correlated.

(3) $sup(xy) > sup(xz)$

In this condition, similar with the previous condition. It is obvious that $sup(xy) \leq sup(y)$, so we can deduce that $Kulc(x, y) > Kulc(x, z)$. If $Kulc(x, y) \leqslant \xi$, the item pair $\{(x, y)\}$ is not positively correlated, we can deduce that $\{(x, z)\}$ is also negative correlated. Otherwise we must calculate $Kulc(x, z)$ to estimate whether the item pair $\{(x, z)\}$ is positively correlated.

In FPMine algorithm, we calculate the interestingness measure of each node with its parents in the FP-tree, at the same time, the interestingness measure value should be recorded in FP-matrix for reusing to reduce the computational overhead of our algorithm.

4. PFPMine: Parallel frequent pattern based mining algorithm

As the volume of dataset gets larger, most of the traditional frequent pattern mining algorithms cannot mining frequent patterns with high efficiency due to huge resource requirement or too much computation consumption. In this section we propose a parallel algorithm to discover the frequent 2-itemsets, called PFPMine. This algorithm parallelizes the FPMine algorithm using MapReduce framework to solve the problem produced by the emerging large-scale dataset.

MapReduce [26] is a parallel and scalable programming model for data intensive applications and large scale data analysis. A MapReduce computation model can be divided into two phase tasks, which are Map task and Reduce task. The map phase splits the input dataset into several dataset shards, and each map task uses a key–value pair as the input, and then a set of intermediate key–value pairs are generated for next phase task use. Then the intermediate values are delivered to the reduce tasks. After shuffling, each reduce task accepts all intermediate pairs associated with a particular key and generates a final set of key–value pairs. In frequent pattern mining area, some researchers proposed parallel frequent pattern mining algorithm to

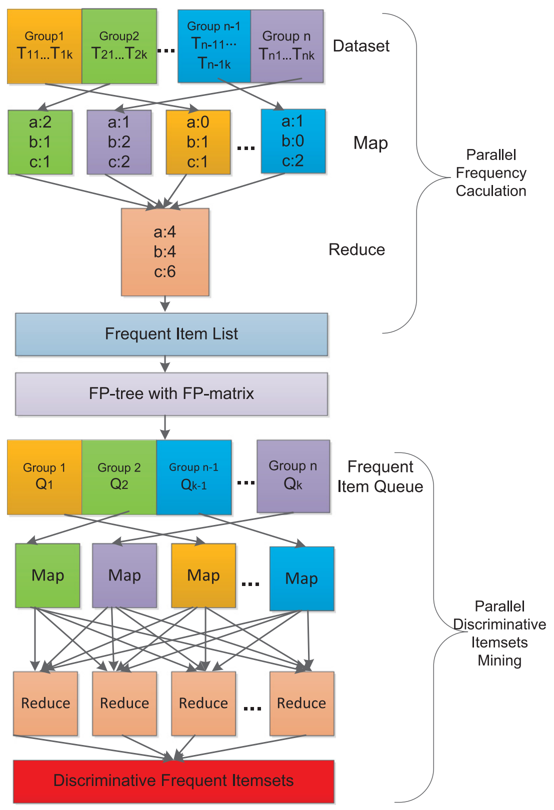


Fig. 3. The overall PFPMine framework.

improve the performance of traditional frequent pattern mining algorithm [27,28].

In light of the characteristics of MapReduce computation model, we design a parallel frequent pattern based algorithm, namely PFPMine. Fig. 3 depicts the framework of PFPMine. PFPMine can be divided into two round MapReduce to parallelize the FPMine algorithm. Specifically, the two round MapReduce can be described as follows.

1. The first MapReduce phase is parallelizing the frequency calculating algorithm. This round divides the workflow log into several shards and then each mapper picks one shard of the workflow log to calculate the local frequency information. Finally, the reducer is performed to gain the global frequency information.
2. The second MapReduce phase is parallelizing the discriminative itemsets mining algorithm. This algorithm takes in the frequent item queue, and divides it into several shards. In this round, each mapper takes in the divided frequent item queue to mine the local discriminative itemsets. Subsequently, several reducers are performed to combine the results which are gained by mapper tasks.

In summary, these two phases parallelize the frequent pattern based mining algorithm for mining the discriminative frequent itemsets. Detailed parallel algorithm is explained as follows:

MapReduce framework is able to accelerate the frequency calculation by its divide-and-conquer programming strategy. In our approach, we apply MayReduce technique to the procedures of item frequency calculation which is the first step of our approach. Details can be found from Algorithm 7. The basic idea is to divide the dataset into several shards, each of which is analyzed by a mapper. A reducer is assigned to aggregate all the intermediate result from mappers and finally obtain the global frequency measurement.

Algorithm 5 can also be parallelized by MapReduce framework. Firstly, the frequent item queue is divided into shards and delivered to the mappers. The mappers invoke BRANCH_SEARCH algorithm searching for frequent itemsets based on FP-tree data structure. Essentially, each mapper performs the searching on one branch of the FP-tree. Finally, all the results from mappers are aggregated by multiple reducers. Such calculation can also be completed in a parallel way. Details can be found from Algorithm 8.

Algorithm 7 Algorithm for Parallel Frequency Calculating

```
MAPPER($CID$, $D_i$)
1: $c_j \leftarrow 0$ for all $j = 1, 2, \ldots, n$
2: for all $a_j \in D_i$ do
3:     $c_j \leftarrow c_j + 1$
4: end for
5: for all $j = 1, 2, \ldots, n$ do
6:     EMIT($a_j$, $c_j$)
7: end for
REDUCER($a_j$, $List[c_j]$)
1: $sup(a_j) \leftarrow 0$
2: for all $c_j \in List[c_j]$ do
3:     $sup(a_j) \leftarrow sup(a_j) + c_j$
4: end for
5: EMIT($a_j$, $sup(a_j)$)
```

Algorithm 8 Algorithm for Parallel Mining Discriminative Itemsets

```
MAPPER($i$, $Q_i$)
1: $\beta_j \leftarrow \emptyset$ for all $j = 1, 2, \ldots, n$
2: while $a_j \leftarrow DeQueue(Q_i)$ do
3:     Let $y \leftarrow a_j.parent, z \leftarrow y.parent$
4:     while $a_j \neq null$ do
5:         while $y \neq Root$ do
6:             if $V_{a_j,y} = null$ then
7:                 $V_{a_j,y} \leftarrow Kulc(a_j, y)$
8:             end if
9:             $\beta_j \leftarrow \beta_j \cup$BRANCH_SEARCH($a_j, y, z, M$)
10:        end while
11:        $a_j \leftarrow a_j.Link$
12:    end while
13: end while
14: for all $j = 1, 2, \ldots, n$ do
15:     EMIT($a_j$, $\beta_j$)
16: end for
REDUCER($a_j$, $List[\beta_j]$)
1: $\beta \leftarrow \emptyset$
2: for all $\beta_j \in List[\beta_j]$ do
3:     $\beta \leftarrow \beta \cup \beta_j$
4: end for
5: EMIT($a_j$, $\beta$)
```

5. Data placement and backup strategy

Due to the frequent itemsets obtained by proposed algorithm, the interacting data entities can be stored into the data centers according to a certain strategy. In this section, a primitive data placement and backup strategy is designed to investigate the benefits of our approach.

Fig. 4 illustrates a representative framework of a cloud workflow system. Typically, most of the data entities are stored in the primary data centers, meanwhile, the cloud workflow system backup the data entities in the secondary data centers. A name

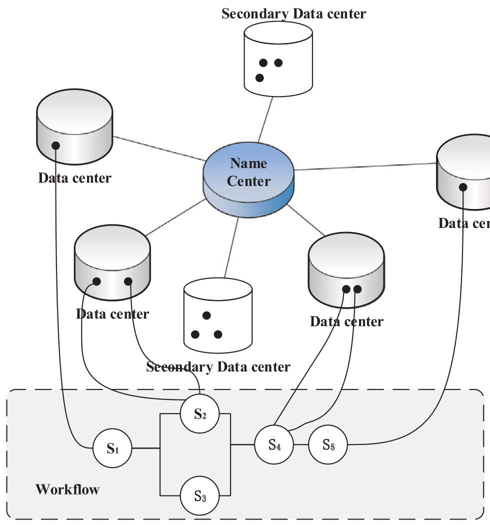


Fig. 4. Framework of cloud workflow system.

center is assigned for managing the storage and backup route of the data entities. It has been universally acknowledged that the transmission and backup of workflow data are quite communicatively costly in geographically dispersed data centers [29], and thus we design a scheme based on the discovery results obtained by our PFPMine algorithm in order to reduce the communication overhead.

Algorithms 9 and 10 show the data placement and backup algorithm. Assuming each data center D_i have a fixed storage space $|D_i|$ and β_j is the set of frequent itemset with the same data item x_j. Firstly, we select a frequent item pair $\beta_{j,k} \subseteq \beta_j$, and then we compute the relationship between the size of the frequent itemset $\beta_{j,k}$ and the data center storage space, if the data center have the capacity to store the frequent itemset, the frequent itemset will be stored into the data center, otherwise another data center with enough storage space will be selected and the frequent itemset will be stored into it. At the same time, we backup the data items into the secondary data center. The storage table ST including the information of data entity placement and backup will be updated accordingly at the same name center.

Algorithm 9 Algorithm for Data Placement and Backup

```
Input: A set of frequent itemset $\beta_j$; Data center $D_i$
Output: Results of data placement and backup
1: Define a storage table: $ST \leftarrow \Phi$
2: for all set of frequent itemsets $\beta_j$ with same data item $x_j$ do
3:     STORE_DATA($D_i$, $\beta_j$)
4:     Backup the frequent itemset into the secondary data center
5:     Update the table $ST$ in the NameCenter
6: end for
```

After discovering frequent itemsets, interacting data entities are placed into data centers according to a certain strategy, and thus the communication overhead is reduced accordingly. We will evaluate the efficiency of our approach based on real-world datasets, the evaluation details can be found in next section.

6. Theoretical analysis and experimental evaluation

In this section, the proposed approach is evaluated from theoretically aspect and experimentally aspect. First, the computational overhead is analyzed from a theoretical viewpoint, and

Algorithm 10 Procedure for STORE_DATA(D_i, β_j)

```
STORE_DATA($D_i$, $\beta_j$)
1: Select a frequent item pair $\beta_{j,k} \subseteq \beta_j$
2: if $|D_i| \geq |\beta_{j,k}|$ then
3:     Store the frequent itemset into the data center
4: else
5:     Search another data center with enough storage space
6:     Store the frequent itemset into the data center
7: end if
8: $|D_i| \leftarrow |D_i| - |\beta_{j,k}|$
9: $\beta_j \leftarrow \beta_j - \beta_{j,k}$
10: STORE_DATA($D_i$, $\beta_j$)
```

then several open-source datasets are used to conduct experimental evaluations.

6.1. Theoretical analysis

This sub-section presents the computational complexity analysis of our proposed algorithms, including constructing FP-tree with FP-matrix, mining discriminative itemsets, and parallel FP-Mine algorithm. Assuming the scale of dataset is n and the number of frequent items in such dataset is m, we analyze the computational complexity of our algorithms as follows.

In Section 3.3, we present the algorithm for constructing FP-tree with FP-matrix. In this algorithm, we have to traverse the dataset before constructing the FP-tree with FP-matrix, so the time complexity of this algorithm is $O(n)$.

The computational complexity of the algorithm for mining discriminative itemsets are analyzed from two aspects. On one hand, we conduct worst-case analysis of the overhead when there is only one branch in the FP-tree. In such situation, every pairs of data items should be analyzed and no intermediate result can be reused. Therefore, the worst-case computational complexity of the algorithm is $O(m^2)$. On the other hand, however, the average complexity is much lower. From the aspect of mean-value analysis, the depth of a regular FP-tree is $\log(m)$, and several intermediate results of *Kulc* value calculations can be reused to further reduce the overhead. In such cases, our algorithm can be executed in $O(\log^2 m)$ time.

In summary, the computational complexity of the FPMine algorithm is $O(n + \log^2 m)$. Comparing with the overhead of traditional FP-growth algorithm which is $O(n + \log m)$, we only pay an additional multiplicative factor $O(\log m)$. Meanwhile, however, more frequent itemsets can be obtained by our approach, which can help significantly especially in cloud workflow environments.

Furthermore, the time complexity of our approach can be further reduced by parallelization which has been introduced in Section 4. On one hand, we split the original dataset to $\sqrt{n}$ subsets and assign $\sqrt{n}$ mappers in the procedure of frequency measurement. Each mapper is able to complete its task in $O(\sqrt{n})$ time, while the reducer is expected to take $O(\sqrt{n})$ time. Therefore, the time complexity of our parallel frequency calculation algorithm is $O(\sqrt{n})$. On the other hand, we split the frequent item queue to $\log m$ shards and assign $\log m$ mappers in the procedure of parallel discriminative mining. Subsequently, $\log m$ reducers are assigned to receive all the results generated by the mappers. In this algorithm, each mapper is able to complete its in $O(\log m)$ time, while each reducer is expected to take $O(\log m)$ time. Therefore, the time complexity of our parallel mining discriminative frequent itemsets algorithm is $O(\log m)$. In conclusion, the overall computational complexity of PFPMine is expected to be $O(\sqrt{n} + \log m)$.

Table 3
Datasets characteristics.

Datasets	#Items	Avg.Length	#Transactions
New Orleans [30]	106	10.93	327
Chicago [30]	159	10.73	676
New York [30]	155	8.47	1200
Retail [31]	16,469	10.31	88,162
Connect [31]	129	43	67,557

6.2. Experimental evaluation

6.2.1. Datasets

In this paper, we use five real-world datasets with different characteristics to evaluate the functionality and performance of proposed algorithm. Table 3 indicates the statistical characteristics of five datasets.

We firstly adopt Entree Chicago Recommendation Data [30] to evaluate the functionality of our approach. This dataset is a real-life one that reflects detailed user preference and order information of different restaurants geographically-dispersed 8 cities of United States from September 1996 to April 1999. Specific semantic information of the datasets is of value to illuminate the effectiveness of our approach. We pick up three groups data with different characteristics comes from different cities to evaluate the performance of our approach, which are New Orleans, Chicago and New York.

For performance evaluation, we choose another dataset [31] because of the following two reasons. Firstly, performance evaluation requires a large volume of testing data especially for scalability analysis, and thus the scales (number of transactions) of datasets we use in this part are much larger than the previous ones. Secondly, the datasets have been widely adopted for the performance evaluation of data mining algorithms, and the experimental results on such datasets should be with reference value when we compare our approach with other traditional data mining algorithms. However, all the data from [31] has been anonymized and preprocessed, and hence is not able to be used for functional validation in the previous part. Among the datasets, we select two of them with different characteristics. The first one is a sparse dataset named “Retail”, which has 88,162 transactions and 16,469 items, whose average length is 10.3 for each transaction. The other one named “Connect” is a dense dataset with 67,557 transactions and 129 items, whose average length is 43 for each transaction.

Before the experimental evaluation, we conduct a brief analysis of the datasets to validate the assumption we have made in the previous parts of this paper. Illustratively, the dataset with the largest scale namely Retail is selected, and its Probability Mass Function (PMF) of data item frequencies is analyzed shown by Fig. 5. Basically, the frequencies of data items conform to Zipf distribution. More specifically, we validate such assumption quantitatively by analyzing its coefficient of determination [32], which has been widely applied to evaluate the goodness of data fitting and prediction. Mathematically, such metric is formulated by (8) as follows.

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i - f_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2} \tag{8}$$

where y_i is the i-th data item of the ground truth, f_i is the i-th predicted data item by the fitting model, and $\bar{y}$ is the mean value of the observed data. The value of R^2 ranges from 0 to 1, where an R^2 of 1 indicates that the fitting model perfectly fits the real data. In our real data based experiments, we obtain $R^2 = 0.90222$ according to its definition, which validates the soundness of the assumption we made for data frequency distribution.

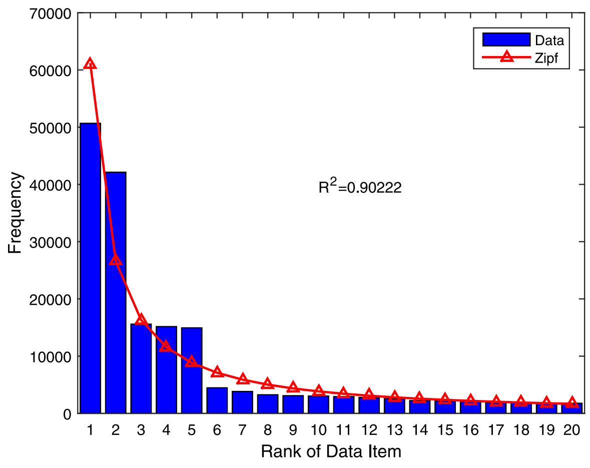


Fig. 5. The PMF of Retail dataset.

6.2.2. Functional evaluation

We conduct experiments on real-life datasets with specific semantic information for frequent itemset mining with our PFPMine algorithm. Table 4 shows part of the results obtained by our approach from the datasets, which reveal some potential relationships of data items. For instance, seafood may be the specialty of the restaurant named “Oyster Bars” in New Orleans since a great many of data pairs {SeaFood, Oyster Bars} have been extracted from the dataset. Chinese in Chicago often have brunch at the restaurants in weekends, which shows a lifestyle of the Chinese people abroad. Moreover, it can be concluded from the data analysis that hamburgers are often ordered in coffee shops in Chicago. In New York, data analytical results show that Americans prefer sea food, hamburgers and bear while Italians prefer pizza. Such results can be with great value for the businessmen for restaurant management and food recommendations.

Furthermore, we conduct the evaluation to illuminate the benefits of our approach for data placement and backup. The experiments are evaluated from two aspects. On one hand, we assume the communication overhead between different data centers is 1 and the communication overhead in a same data center is zero, and compare the overall overhead of our scheme and FP-growth. On the other hand, we assume the communication overheads between data centers are uniformly distributed over random values ranging from 0 to 1. After 30 random evaluations, the results are shown in Fig. 6.

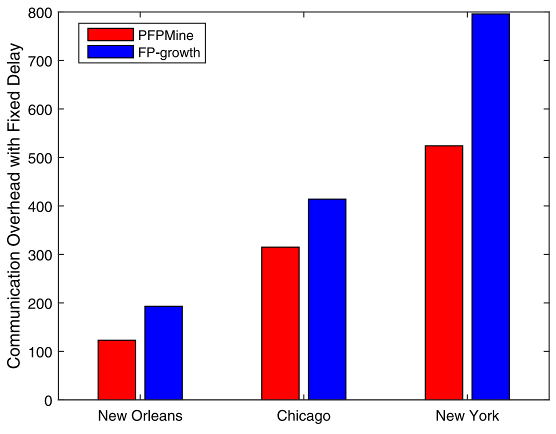
Specifically, Fig. 6(a) and 6(b) show the evaluation results using PFPMine and FP-growth on three different datasets with different delays, which indicate the communication overhead of PFPMine algorithm is lower than that of the classical FP-growth algorithm. As the size of datasets become larger, the results seem to be more pronounced. Fig. 6(c) illustrates the PDF of communication overhead differentials between PFPMine algorithm and FP-growth on three different datasets. It is obvious that the communication overhead differentials have different bounds on different datasets, which are bounded from 25.5 to 44.33, 34.03 to 64.47, and 116.58 to 157.92 with high probability on New Orleans, Chicago, and New York, respectively.

6.2.3. Performance evaluation

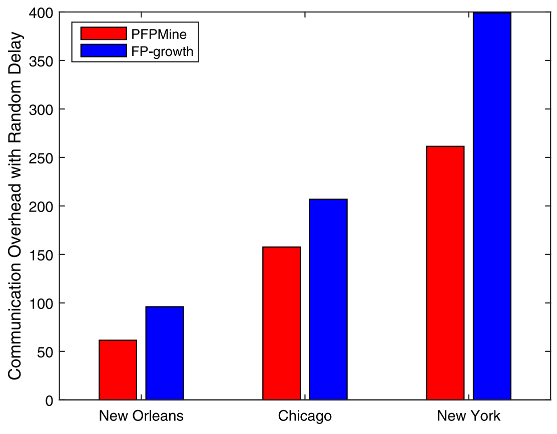
We evaluate the efficiency of our algorithm and the traditional FP-growth on real-world datasets with ten virtual nodes, where each node is equipped with Intel Core i7 Quad-Core CPU and 8GB memory. We implemented the FP-growth, FPMine and PFPMine algorithms and conduct experiments to evaluate the scalability of our approach. The experimental results in terms of the running

Table 4
Selecting mining results.

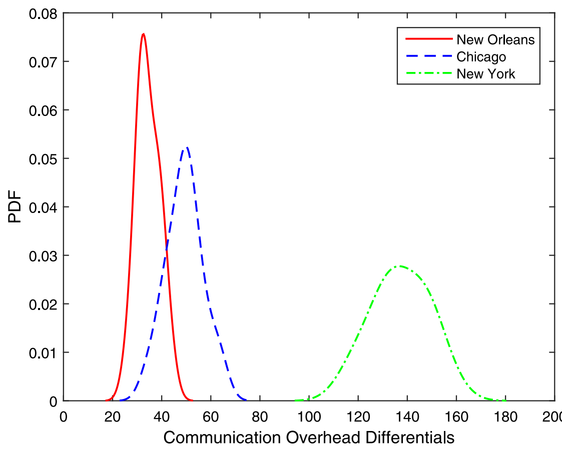
Datasets	Frequent Itemsets
New Orleans	{Oyster Bars, SeaFood}, {Hamburgers, Pizza}
Chicago	{Chinese, Weekend Brunch}, {Hamburgers, Coffee Shops}
New York	{American, SeaFood}, {Italian, Pizza}, {American, Hamburgers & Bear}



(a) Communication Overhead with Fixed Delay



(b) Communication Overhead with Random Delay



(c) PDF of Communication Overhead Differentials

Fig. 6. Communication overhead.

time of FP-growth and FPMine are obtained by straightforward measurement, while the parallelized parts of PFPMine are measured with ten virtual homogeneous nodes. In order to estimate the error caused by the dynamic changes of experimental environment, we evaluate the running time for 30 times, and compute the average of the values.

Fig. 7 shows the execution time of PFPMine, FPMine, and FP-growth on real-world dataset with different scales. Before parallelization, the results of FP-growth and FPMine indicate both of their running times are proportional to the scale of datasets (number of transactions), which validates the theoretical analysis in Section 6.1. Comparing with FP-growth, we pay only a multiplicative factor for making our approach able to extract fine-grained frequent item pairs rather than frequent patterns, which has been proved to bring significant advantages to data placement. Furthermore, after parallelized by the MapReduce framework, the performance of our approach is significantly improved. At the cost of only a small quantity of computing resources which are commonly plentiful in cloud environments, our approach can be completed in a reasonably small time of $O(\sqrt{n})$. Fig. 7(a) and 7(b) illustrate that our approach is able to perform well with high efficiency for both sparse datasets and dense datasets, which validate the efficacy of our approach in different scenarios. The results show that PFPMine can reduce the running time by at least 25%, and preforms with significantly higher efficiency than FP-growth approach.

To indicate the efficiency of PFPMine comparing with FPMine, we conduct the experiments for speedup evaluation of PFPMine based on real-life dataset. Fig. 8 illustrates the speedup of the PFPMine when we scale up the size of real-life dataset. Fig. 8(a) shows the speedup goes up slightly with the number of transaction increases from 50,000 to 80,000 on Retail dataset. Fig. 8(b) shows the speedup goes up slightly with the number of transactions increases from 40,000 to 60,000 on Connect dataset. All of

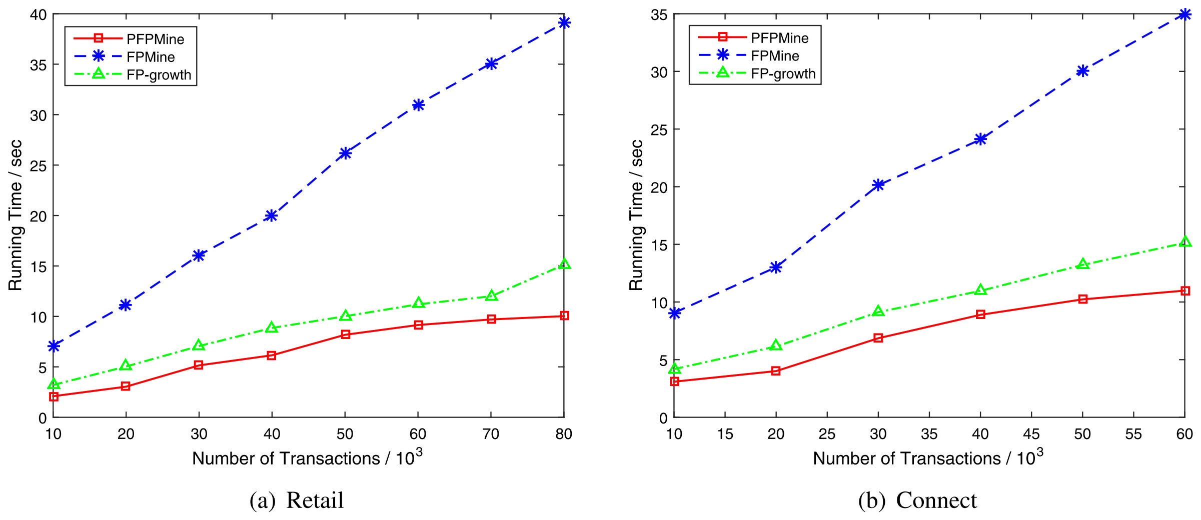


Fig. 7. Performance analysis.

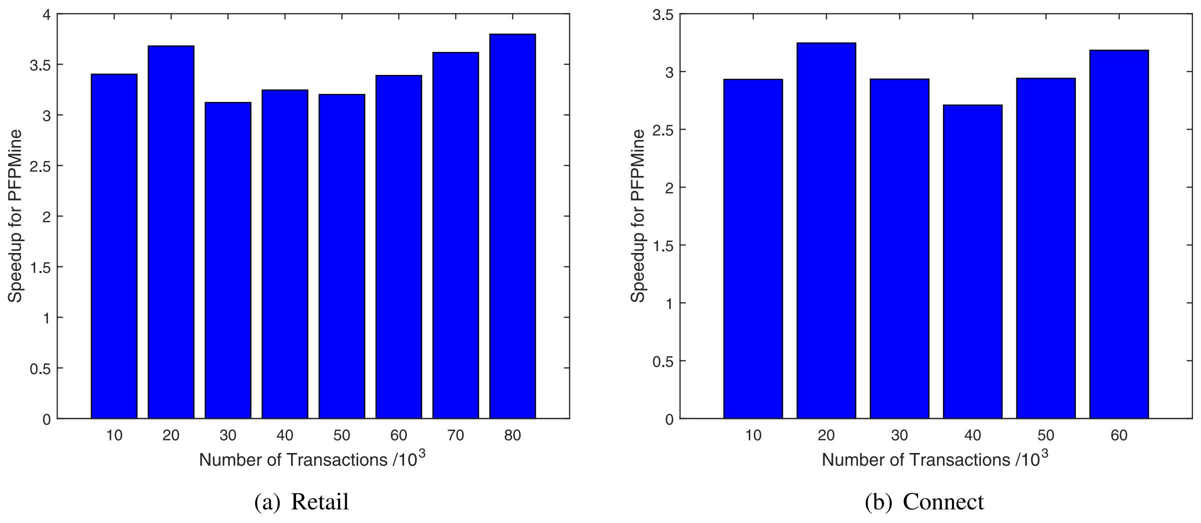


Fig. 8. Speedup analysis.

the results show that PFPMine can improve the performance of FPMine significantly both on sparse and dense datasets.

7. Related work

7.1. Data-intensive workflow management

In this paper, business process modeling is the foundation of our research work, which formulates business processes (workflow) by formal languages understandable for computers. Traditional business process modeling approaches focus on the process of the activities, named process-centric model. It has been researched for over a decade. Some process modeling language and workflow system have been developed and widely adopted. Traditional process-centric modeling approach can express the procedure of activities clearly, but it rarely considers the data in business process, and thus cannot formulate the operations (read, write, add, remove, *etc.*) of global data clearly during the procedures of the design and implementation for the data-intensive systems. With the emergence of data-intensive applications and systems, Data-aware modeling approach (also called data-centric modeling) have been proposed to fill this gap [33,34]. In our previous work, we have proposed a novel data-aware business process modeling approach and developed the prototype system [19], based on which, the main thrust of this paper is to propose an efficient scheme of data entity discovery for providing fundamental theoretical support for data placement and backup in cloud workflow systems.

In data-aware business process, data entity records the core information for business operations. Discovering the artifacts is one of the most important challenges in this area. Recently, several approaches have been proposed by researchers [35–37]. However, most of the existing approaches neglected the potential relationships between data entities, which may result in the drawback for data placement in multiple processes environments. In this paper, we make an attempt at discovering the interacting data entities in data-intensive workflows to provide the fundamental support for data placement and backup.

7.2. Data management for cloud workflows

In order to utilize the resource with high efficiency, the most effective method is to deploy the workflow into the cloud environment. Data management for cloud workflows is a very important and challenging problem, which includes data placement and data backup. It is acknowledged that cloud workflow system is designed for different communities for collaborative work, where distributed applications need to be executed across different data centers. The data movement across different data centers may cost numerous time and resources, and some researches focused on placement strategies for cloud system. For instance, Yuan et al. [38] proposed a strategy for data placement in cloud based scientific workflows, which adopts a matrix based k-means clustering approach to store the data into data centers. Jaradat et al. [39] presented a novel workflow partitioning and deployment approach, which moves the workflow computation towards the data to obtain the optimal performance results. Kumar et al. [40] minimize the average query span using a workload-aware data placement algorithm for cloud data management systems. Sun et al. [41] presented an adaptive data placement approach to reduce data access cost, which places the data within a data staging area dynamically and adaptively. Zhang et al. [42] presented a clustering-based algorithm for data placement to reduce the workload of data transfer between several data centers. Besides data placement for cloud workflows, some researches also presented some approach on data backup in cloud environment [43–45]. But most of them mainly focused on the schema of data backup, ignoring studying the importance level and inter-relationship of large-scale data entities.

In summary, although some contributions have been proposed, they rarely consider the frequent itemsets in business process, which ignored the importance information of data items. In certain circumstances, they may result in useless or unimportant data items during the procedure of data backup, which will bring more workload to data centers. Therefore, this paper studies such issues from a very different angle, trying to studying the importance and frequency levels of data items. Our approach can provide fundamental theoretical support for data management (including data placement and backup) for cloud workflows.

7.3. Frequent pattern discovery

Frequent pattern discovery is a research hotspot in the field of data mining in recent years, which was first proposed by Agrawal et al. for market basket analysis [46]. Since then, abundant literatures have been dedicated to this research, and significant progresses have been made. Apriori is the most typical algorithm proposed by Agrawal and Srikant, which observed an interesting downward closure property from super market shopping [47]. Afterwards, some extensive studies on the improvements or extensions of Apriori have been proposed [48,49]. FP-growth is another algorithm for frequent pattern mining, which mines the complete set of frequent patterns without candidate generation [16]. Some alternatives or extensions of FP-growth can improve the performances of the algorithm [50,51], but they cannot discover the fine-grained frequent 2-itemsets directly.

We note that the frequent pattern mining is also useful to other data analysis and mining tasks [52,53] which is similar to the issue of this paper. On the other hand, however, existing approaches cannot be directly applied to data entities discovery due to the characteristics of business processes in cloud workflows. Although some Apriori-like algorithms can discover the frequent 2-itemsets by join arithmetic [17], which can generate huge useless candidate sets and cost numerous time and resources. The FP-growth and its extensions cannot discover the fine-grained frequent 2-itemsets directly, which consume about 80% computational resources for traversing FP-tree. In this paper, we present an approach to mine the fine-grained frequent item pairs other than the frequent patterns.

8. Conclusion

In this paper, we present a frequent pattern based approach to discover interacting data entities in cloud workflows. Our approach first determines the minimum support threshold automatically, and then constructs FP-tree with FP-matrix, which stores the support count and interestingness measure values of item pairs. With such data structure construction, the discriminative frequent 2-itemsets can obtained with high efficiency. Furthermore, the algorithm is parallelized by MapReduce framework to reduce its overhead from $O(n + \log^2 m)$ to $O(\sqrt{n} + \log m)$. Finally, the efficiency of the algorithms are evaluated by both theoretical analysis and real-world data based experimental evaluations.

The main thrust of this paper is to propose an efficient scheme of data entity discovery for providing fundamental theoretical support for data placement and backup in cloud workflow systems. The motivation is to understand the importance level and inter-relationship among data by studying the original data operations in workflows. We also designed a primitive data placement and backup strategy based on our data entity discovery approach. Please note that the detailed design and analysis of data placement and backup schemes are out of the scope of this paper. In the future, we plan to guide the design the detail data placement strategies for cloud workflows using the results obtained by our approach. We also noticed that, due to the limitation of hardware, we evaluate the communication overhead of our algorithm by real-world datasets in laboratory environments. We will develop the data-intensive system and record the real-life data to conduct the experiment on real homogeneous nodes to investigate the efficiency of our approach.

CRediT authorship contribution statement

Yuze Huang: Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Jiwei Huang:** Methodology, Writing - review & editing. **Cong Liu:** Writing - review & editing. **Chengning Zhang:** Software, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the Project of Science and Technology Research Program of Chongqing Education Commission of China (No. KJQN201900708), Beijing Natural Science Foundation(No. 4202066), National Natural Science Foundation of China (Nos. 61972414, 61902222). Taishan Scholars Program of Shandong Province (No. tsqn201909109), and Fundamental Research Funds for Central Universities (No. 2462018YJRC040)

References

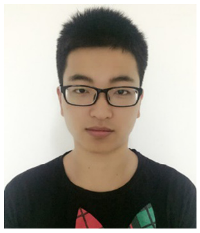
[1] I. Gorton, P. Greenfield, A. Szalay, R. Williams, Data-intensive computing in the 21st century, Computer 41 (4) (2008) 30–32.
[2] Z.T. Can, M. Demirbas, Smartphone-based data collection from wireless sensor networks in an urban environment, J. Netw. Comput. Appl. 58 (2015) 208–216.
[3] Y. Peng, J. Li, S. Park, K. Zhu, M.M. Hassan, A. Alsanad, Energy-efficient cooperative transmission for intelligent transportation systems, Future Gener. Comput. Syst. 94 (2019) 634–640.
[4] M. Elshenawy, B. Abdulhai, M. El-Darieby, Towards a service-oriented cyber-physical systems of systems for smart city mobility applications, Future Gener. Comput. Syst. 79 (2018) 575–587.

[5] D. Calvanese, G. De Giacomo, M. Montali, Foundations of data-aware process analysis: A database theory perspective, in: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2013), 2013, pp. 1–12.
[6] A. Nigam, N.S. Caswell, Business artifacts: An approach to operational specification, IBM Syst. J. 42 (3) (2003) 428–445.
[7] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616.
[8] K. Kanagaraj, S. Swamynathan, Structure aware resource estimation for effective scheduling and execution of data intensive workflows in cloud, Future Gener. Comput. Syst. 79 (2018) 878–891.
[9] L. Zeng, B. Veeravalli, A.Y. Zomaya, An integrated task computation and data management scheduling strategy for workflow applications in cloud environments, J. Netw. Comput. Appl. 50 (2015) 39–48.
[10] T. Xie, SEA: A striping-based energy-aware strategy for data placement in RAID-structured storage systems, IEEE Trans. Comput. 57 (6) (2008) 748–761.
[11] R. Tudoran, A. Costan, G. Antoniu, Overflow: Multi-site aware big data management for scientific workflows on clouds, IEEE Trans. Cloud Comput. 4 (1) (2016) 76–89.
[12] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, D. Yuan, A genetic algorithm based data replica placement strategy for scientific applications in clouds, IEEE Trans. Serv. Comput. 11 (2018) 727–739.
[13] X. Li, L. Zhang, Y. Wu, X. Liu, E. Zhu, H. Yi, F. Wang, C. Zhang, Y. Yang, A novel workflow-level data placement strategy for data-sharing scientific cloud workflows, IEEE Trans. Serv. Comput. 12 (3) (2019) 370–383.
[14] B. Yu, J. Pan, Location-aware associated data placement for geo-distributed data-intensive applications, in: 2015 IEEE Conference on Computer Communications (INFOCOM 2015), 2015, pp. 603–611.
[15] I. Casas, J. Taheri, R. Ranjan, L. Wang, A.Y. Zomaya, A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems, Future Gener. Comput. Syst. 74 (2017) 168–178.
[16] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of International Conference on Management of Data (SIGMOD 2000), 2000, pp. 1–12.
[17] C.C. Aggarwal, J. Han, Frequent Pattern Mining, Springer, 2014.
[18] W.M. Aalst, Process Mining: Data Science in Action, Springer, 2016.
[19] Y. Huang, J. Huang, B. Wu, J. Chen, Modeling and analysis of data dependencies in business process for data-intensive services, China Commun. 14 (10) (2017) 151–163.
[20] G. Grahne, J.F. Zhu, Fast algorithms for frequent itemset mining using FP-trees, IEEE Trans. Knowl. Data Eng. 17 (10) (2005) 1347–1362.
[21] H. Yun, D. Ha, B. Hwang, K.H. Ryu, Mining association rules on significant rare data using relative support, J. Syst. Softw. 67 (3) (2003) 181–191.
[22] K. Wang, Y. He, J.W. Han, Pushing support constraints into association rules mining, IEEE Trans. Knowl. Data Eng. 15 (3) (2003) 642–658.
[23] A. Corral, G. Boleda, R. Ferrer-i Cancho, Zipf's law for word frequencies: Word forms versus lemmas in long texts, PLoS One 10 (7) (2015) 1–23.
[24] L.A. Adamic, Zipf, power-laws, and pareto-a ranking tutorial, 2000, available at http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html.
[25] T. Wu, Y. Chen, J. Han, Re-examination of interestingness measures in pattern mining: a unified framework, Data Min. Knowl. Discov. 21 (3) (2010) 371–397.
[26] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, in: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation (OSDI 2004), 2004, pp. 107–113.
[27] H. Li, Y. Wang, D. Zhang, M. Zhang, E.Y. Chang, PFP: Parallel FP-growth for query recommendation, in: Proceedings of ACM Conference on Recommender Systems (Recsys 2008), pp. 107–114.
[28] Y. Xun, J. Zhang, X. Qin, Fidoop: Parallel mining of frequent itemsets using mapreduce, IEEE Trans. Syst. Man Cybern. 46 (3) (2016) 313–325.
[29] E. Deelman, A. Chervenak, Data management challenges of data-intensive scientific workflows, in: Proceeding of IEEE International Symposium on Cluster Computing and the Grid, 2008, pp. 687–692.
[30] R. Burke, Entree chicago recommendation data, 2000, available at http://infolab.cs.uchicago.edu/entree.
[31] B. Goethals, M.J. Zaki, Frequent pattern mining dataset repository, 2003, available at http://fimi.ua.ac.be/data/.
[32] J.L. Devore, Probability and Statistics for Engineering and the Sciences, ninth ed., Brooks/Cole Publishing Company, 2015.
[33] E. Damaggio, R. Hull, R. Vaculin, On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles, Inf. Syst. 38 (4) (2013) 561–584.
[34] Y. Sun, J. Su, J. Yang, Universal artifacts: A new approach to business process management (BPM) systems, ACM Trans. Manag. Inf. Syst. 7 (3) (2016) 3:1–3:26.
[35] E.H.J. Nooijen, B.F. Van Dongen, D. Fahland, Automatic discovery of data-centric and artifact-centric processes, in: Proceeding of International Workshop on Business Process Management (BPM 2012), 2012, pp. 316–327.
[36] V. Popova, D. Fahland, M. Dumas, Artifact lifecycle discovery, Int. J. Coop. Inf. Syst. 24 (1) (2015) 1–44.
[37] X. Lu, M. Nagelkerke, D. van de Wiel, D. Fahland, Discovering interacting artifacts from ERP systems, IEEE Trans. Serv. Comput. 8 (6) (2015) 861–873.
[38] D. Yuan, Y. Yang, X. Liu, J. Chen, A data placement strategy in scientific cloud workflows, Future Gener. Comput. Syst. 26 (8) (2010) 1200–1214.
[39] W. Jaradat, A. Dearle, A. Barker, Workflow partitioning and deployment on the cloud using orchestra, in: Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2014), 2014, pp. 251–260.
[40] K.A. Kumar, A. Quamar, A. Deshpande, S. Khuller, SWORD: workload-aware data placement and replica selection for cloud data management systems, VLDB J. 23 (6) (2014) 845–870.
[41] Q. Sun, T. Jin, M. Romanus, H. Bui, F. Zhang, H. Yu, H. Kolla, S. Klasky, J. Chen, M. Parashar, Adaptive data placement for staging-based coupled scientific workflows, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2015), 2015, pp. 65:1–65:12.
[42] J. Zhang, M. Wang, J. Luo, F. Dong, J. Zhang, Towards optimized scheduling for data-intensive scientific workflow in multiple datacenter environment, Concurr. Comput.-Pract. Exp. 27 (18, SI) (2015) 5606–5622.
[43] J. Lin, C. Chen, J.M. Chang, QoS-Aware data replication for data-intensive applications in cloud computing systems, IEEE Trans. Cloud Comput. 1 (1) (2013) 101–115.
[44] M. Wang, L. Zhu, Z. Zhang, Risk-aware intermediate dataset backup strategy in cloud-based data intensive workflows, Future Gener. Comput. Syst. 55 (2016) 524–533.
[45] Y. Ebadi, N.J. Navimipour, An energy-aware method for data replication in the cloud environments using a tabu search and particle swarm optimization algorithm, Concurr. Comput.: Pract. Exper. 31 (1) (2019) 1–10.
[46] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD 93), 1993, pp. 207–216.
[47] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 94), 1994, pp. 487–499.
[48] F. Geerts, B. Goethals, J. Van Den Bussche, A tight upper bound on the number of candidate patterns, in: Proceedings of 1st IEEE International Conference on Data Mining (ICDM 2001), pp. 155–162.
[49] S.K. Yun, N. Rountree, Finding sporadic rules using apriori-inverse, in: Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2005), 2005, pp. 97–106.
[50] Y. Jiang, M. Zhao, C. Hu, L. He, H. Bai, J. Wang, A parallel FP-growth algorithm on world ocean atlas data with multi-core CPU, J. Supercomput. 75 (2) (2019) 732–745.
[51] J. Ragaventhiran, M.K.K. Devi, Map-optimize-reduce: CAN tree assisted FP-growth algorithm for clusters based FP mining on hadoop, Future Gener. Comput. Syst. 103 (2020) 111–122.
[52] N. Kashmar, M. Atieh, Mining frequent patterns to identify vertical handover parameters in cellular networks, J. Amb. Intell. Hum. Comput. 9 (1) (2018) 31–42.
[53] Y. Chen, P. Yuan, M. Qiu, D. Pi, An indoor trajectory frequent pattern mining algorithm based on vague grid sequence, Expert Syst. Appl. 118 (2019) 614–624.

Yuze Huang is an assistant professor in the School of Information Science and Engineering at Chongqing Jiaotong University. He received the Ph.D. degree in Computer Science and Technology from Beijing University of Posts and Telecommunications in 2018, the M.Eng. degree from Kunming University of Science and Technology in 2013 and the B.Sc. degree from Hangzhou Dianzi University in 2008. His main research interests include services computing and edge computing. He is a member of the IEEE and ACM. Email: huangyz@cqjtu.edu.cn

Jiwei Huang is a professor and the Dean of the Department of Computer Science and Technology at China University of Petroleum-Beijing. He is the director of Beijing Key Laboratory of Petroleum Data Mining. He received the Ph.D. degree and B.Eng. degree both in computer science and technology from Tsinghua University in 2014 and 2009, respectively. He was a visiting scholar at Georgia Institute of Technology. His research interests are in services computing and performance evaluation. He is a member of the IEEE and ACM. Email: huangjw@cup.edu.cn

Cong Liu is a professor in the School of Computer Science and Technology at Shandong University of Technology. He received the B.S. and M.S. degrees in computer software and theory from the Shandong University of Science and Technology, Qingdao, China, in 2013 and 2015, respectively, and the Ph.D. degree in computer science and information systems from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 2019. His current research interests include services computing, business process management, process mining, and Petri nets.



Chengning Zhang is a data engineer in Grab company in Singapore. He received master degree in 2017 from Singapore Management University in Singapore and the B.Eng. degree in 2016 from Beijing University of Posts and Telecommunications in China. His research interests include workflow management and data mining.