IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INNOVATION AND APPLICATION OF INTELLIGENT PROCESSING OF DATA, INFORMATION AND KNOWLEDGE AS RESOURCES IN EDGE COMPUTING

Received August 10, 2019, accepted August 26, 2019, date of publication September 3, 2019, date of current version October 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939162

Detecting Difference Between Process Models Using Edge Network

JIAXING WANG[1], (Member, IEEE), BIN CAO[1], (Member, IEEE), XI ZHENG[2], (Member, IEEE), DAPENG TAN[3], (Member, IEEE), AND JING FAN[1], (Member, IEEE)

[1]College of Computer Science and Software Engineering, Zhejiang University of Technology, Hangzhou 310023, China
[2]Department of Computing, Macquarie University, Sydney, NSW 2109, Australia
[3]Key Laboratory of E&M, Ministry of Education and Zhejiang Province, Zhejiang University of Technology, Hangzhou 310023, China

Corresponding author: Jing Fan (fanjing@zjut.edu.cn)

This research was partially supported by National Key Research & Development Program of China under Grant 2018YFB1402802.

ABSTRACT Process difference detection has played an important role in business process management for enterprise applications. However, the business processes are becoming more and more complex, involving a wide spectrum of tasks and different types of execution orders among these tasks, such as sequential, parallel, loop and conditional order. Thus, there is a need to detect difference between two process models efficiently. To meet this requirement, we use a difference detection framework for process models based on edge computing, where the edges can perform the task of difference detection between two process models, and the difference detection results can be aggregated to the cloud center. Most existing approaches detect process difference based on one feature of a process model, while a process model actually contains multiple features such as structure, behavior, and performance. In this paper, we propose an approach that can detect both structural and behavioral differences between two process models, which provides two aspects of difference information to process analysts and these kinds of insights are helpful to improve the original process model with low-cost and high-efficiency. First, we transform the process models into their corresponding task-based process structure trees (TPSTs) and assign each TPST node a feature vector based on the one-hot encoding. Then, the common key structure of two process models is extracted by comparing the feature vectors of nodes. Finally, the structural and behavioral differences are displayed in terms of this common key structure. Both the case study and efficiency study are provided to show the practicality of the proposed approach.

INDEX TERMS Process model, process difference, edge computing, one-hot encoding, key structure.

I. INTRODUCTION

Detecting difference between two business process models is one of the important services in business process management. It can be used in the following scenarios: (1) A business process has more than one versions, and these different versions may be modeled by diverse subsystems and various process designers at the same time. Finally, these different versions of process models need to be compared and merged into a new one, which contains all the changes proposed by all designers [1]. (2) Changes to the process model may introduce data inconsistencies, which needs to be detected, handled and tracked according to the results of process difference detection [2], [3].

Most existing methods focus on only one feature of a process model, such as structure and behavior. The structure-based methods [4]–[6] represent the structural differences as a set of edit operations that can transform one process model into another. However, the problem of computing the graph edit distance between two process models is at least as hard as subgraph isomorphism, which is NP-hard. The behavior-based methods detect the execution trace differences between two process models [7]–[9], while the state space explosion will happen once process models contain many parallel patterns.

The associate editor coordinating the review of this article and approving it for publication was Honghao Gao.

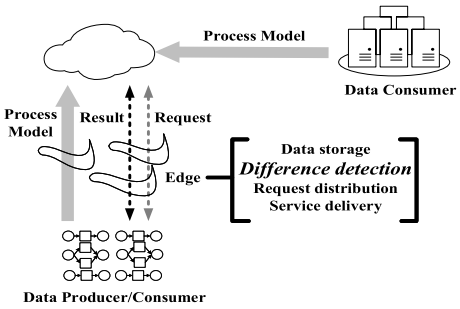


FIGURE 1. Process difference detection framework based on edge network.

Since the process models have become more complex over time, including a wide spectrum of tasks and different types of execution orders among these tasks, there is a need to efficiently detect difference between two process models. In this way, the process designer can quickly improve the existing process model according to the difference to meet the frequent changes of custom and market demands. However, the mentioned existing techniques cannot meet this new requirement, it is because these methods involve long computations of graph edit distance or execution trace which is extremely extensive. To address this problem, we use a process difference detection framework based on edge network to improve the computational efficiency. Nowadays, edge computing are widely applied, where services are provided in a decentralized architecture, and processed by a local computer or server [10].

Figure 1 shows the two-way computing streams in edge computing, where the things are both producers and consumers. At the edge, the things can not only request service from the cloud, but also perform the computing tasks from the cloud. The edges can perform data storage, difference detection between process models, distribute request and delivery service from cloud to users [11]. Thus, the process difference results generated from edge nodes can be aggregated to the cloud center in real time. In this way, the center has global monitoring for all process models, and the process designers can reallocate the process resources and improve the execution efficiency according to the process difference results.

In this paper, we propose an approach to detect differences between two process models using edge network. Both structure and behavior are considered, it is because a process model has multiple features, such as structure, behavior, cost [12], and QoS [13]–[18], and considering more features of a process model will provide more aspects of difference information to process designers. In this way, more useful insights about where and how two process models differ can be uncovered, and these kinds of insights are helpful to improve the original process model and obtain a cost-reduction and high-efficiency process model.

Since the structure is the most important feature of a process model, we use the key structure to abstract the structure of a process model. A key structure describes what kinds of control flow patterns the process model contains and these patterns are in which execution orders. The common key structure of two process models abstracts their common parts, while the different parts are still unknown. To detect the structural as well as behavioral differences between two process models, we introduce one-hot encoding [19] that is popular in machine learning to encode each task node. For the control flow patterns, we design several strategies to encode them based on the one-hot codes. In this way, each node or control flow pattern is represented by a feature vector, and we can get the differences by comparing the feature vectors, which can be displayed beyond their common key structure.

Next, we highlight our contributions as follows:

- A process difference detection framework based on edge network is used to improve the computational efficiency.
- One-hot encoding is used for encoding the task nodes, and three strategies are designed to encode the control flow patterns in a process model.
- Both case study and efficiency study are conducted to show the practicality of the presented approach.

The rest of this paper is organized as follows. Section II sets the stage for the concepts used in this paper. The implementation and experimental evaluation are presented in Section III and IV, respectively. Section V reviews the related work, and Section VI concludes this paper.

II. PRELIMINARIES

This section presents a set of preliminaries that are important to set the stage for understanding this paper and its vision. In particular, we first present the process modeling and the task-based process structure tree, which are the basis of our work. Then we introduce the one-hot encoding.

A. PROCESS MODELING

A process consists of a set of tasks and their relations to reach a goal. The structure of a process can be modeled as a directed graph that can be denoted as a tuple $P = (T, G, E, S, O)$, where T, G, E, S and O are task node set, gateway node set, edge set, start node and end node, respectively, and $G = \{And\text{-}split, Xor\text{-}split, loop\text{-}split, And\text{-}join, Xor\text{-}join, loop\text{-}join\}$.

Taking $Process_2$ in Figure 2 as an example, its task node set is $\{A, B, C, D, E, F\}$, the gateway node set is $\{And\text{-}split, Xor\text{-}split, And\text{-}join, Xor\text{-}join\}$ that consists of parallel and conditional patterns. Its start and end nodes are *start* and *end*, and it contains 13 edges, such as $start \rightarrow A$ and $E \rightarrow F$.

There are four common control flow patterns in a process model: sequential, parallel, conditional and loop patterns [20], which are labeled as *Sequence*, *And*, *Xor*, and *Loop*, respectively. The characteristics of these four common control flow patterns are:

- **Sequential pattern.** Each task node in the sequential pattern has exactly one incoming arc and one outgoing arc.
- **Conditional pattern.** A conditional pattern that starts from an *Xor-split* node and ends at an *Xor-join* node

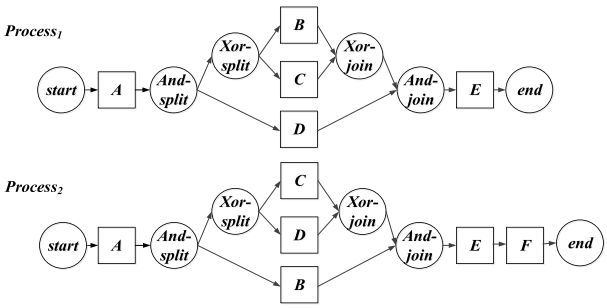


FIGURE 2. Two process models with conditional and parallel patterns.

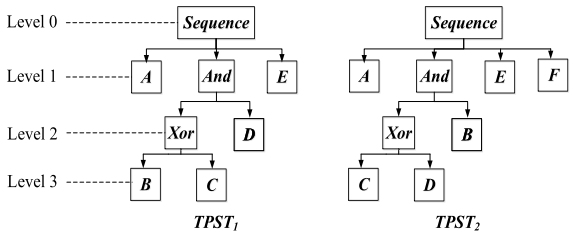


FIGURE 3. Two task-based process structures (TPSTs).

has multiple branches, and only one of the branches is allowed to be selected for execution.

- **Parallel pattern.** A parallel pattern starts from an *And-split* and ends at an *And-join*, and all the task nodes in it are executed simultaneously or in any order.
- **Loop pattern.** A loop pattern has a single entry and exit point [21], which starts from a *Loop-split* node and ends at a *Loop-join* node, and the nodes in it are repeatedly executed.

As shown in Figure 2, there are two process models that contain three kinds of control flow patterns, i.e., sequential, conditional and parallel patterns. Taking $Process_2$ as an example, the highest abstraction level of $Process_2$ is a sequential pattern, which starts from the *start* node and ends at the *end* node. $Process_2$ consists of task A, the parallel pattern *And*, task E and F, which are sequentially executed. The *And* pattern contains a *Xor* pattern and task B, where the task C and D form the *Xor* pattern.

B. TASK-BASED PROCESS STRUCTURE TREE

A process model can be transformed into a task-based process structure tree (TPST) [5], which is a variant of a process structure tree [22]. The features of a TPST are listed in the following: (1) There are four types of gateway nodes in a TPST: *Sequence*, *Loop*, *Xor* and *And*, which correspond to the sequential, loop, conditional and parallel pattern, respectively. (2) The leaf nodes and non-leaf nodes of a TPST separately represent the task nodes and the control flow patterns of its corresponding process model.

Taking the two TPSTs $TPST_1$ and $TPST_2$ in Figure 3 as an example, they are transformed from two process models in Figure 2. The leaf nodes of $TPST_1$ are the task nodes of $Process_1$, i.e., $\{A, B, C, D, E\}$, and its non-leaf nodes are $\{Sequence, And, Xor\}$. The root node of $TPST_1$ is *Sequence*, which shows that the highest abstraction level of $Process_1$ is a sequential pattern.

C. ONE-HOT ENCODING

Given n different states, the one-hot encoding of the k-th state is a vector of length n with a single high bit 1 at the k-th place, and all the other bits are low 0. The definition of one-hot [19] is listed as follows:

Theorem 1: One-Hot. Let set $A = \{0, 1\}$, $N \in \mathbf{Z}^+$, $N > 1$ and $n \in \mathbf{Z}^+$, such that $1 \leq n \leq N$. The n-th one-hot vector of N bits is defined as vector $oh^n \in A^N$, where $oh^n_n = 1$ and $oh^n_i = 0, \forall\, i \neq n, 1 \leq i \leq N$.

For example, there is a property "name" with three values, i,e., {"Jason", "Lucy", "Emmy"}. With one-hot encoding, each value becomes a one-hot vector with size of 3, where only one element in each vector is non-zero and the non-zero elements of all vectors are in different positions. Thus, "Jason", "Lucy" and "Emmy" are encoded as "[1,0,0]", "[0,1,0]" and "[0,0,1]", respectively.

III. IMPLEMENTION

In this section, we introduce the implementation of the presented approach. Given two process models, the main idea is to extract their common key structure based on one-hot encoding, and both structural and behavioral differences can be displayed beyond this common key structure. There are three consecutive phases, namely, *one-hot encoding*, *common key structure extraction*, and *difference detection*. These three phases are described in details in the rest of this section. For illustration, we use two process models shown in Figure 4 throughout this whole section.

A. PHASE 1: ONE-HOT ENCODING

The inputs of this phase are two process models, the outputs are their corresponding encoded TPSTs.

Main Idea: Given two TPSTs transformed from two process models, we use a one-hot code, which is also regarded as a feature vector, to represent a TPST leaf node, and the feature vector of a TPST non-leaf node is obtained based on its child nodes' feature vectors. Whether two TPST nodes can be mapped or not is determined by comparing their feature vectors. In this way, the common key structure of two process models is created based on the totally mapped nodes, and both structural and behavioral differences can be easily detected based on the unmapped nodes and partially mapped nodes.

Algorithm: Algorithm 1 gives the pseudo-code for the first phase. First, we transform two process models into their corresponding TPSTs (Line 1). In this way, the task nodes and the control flow patterns of a process model can be explicitly presented by leaf nodes and non-leaf nodes in a TPST. Next, this phase mainly performs two tasks: one-hot encoding for TPST leaf nodes and encoding for non-leaf nodes. With regard to task nodes, we first get the union of

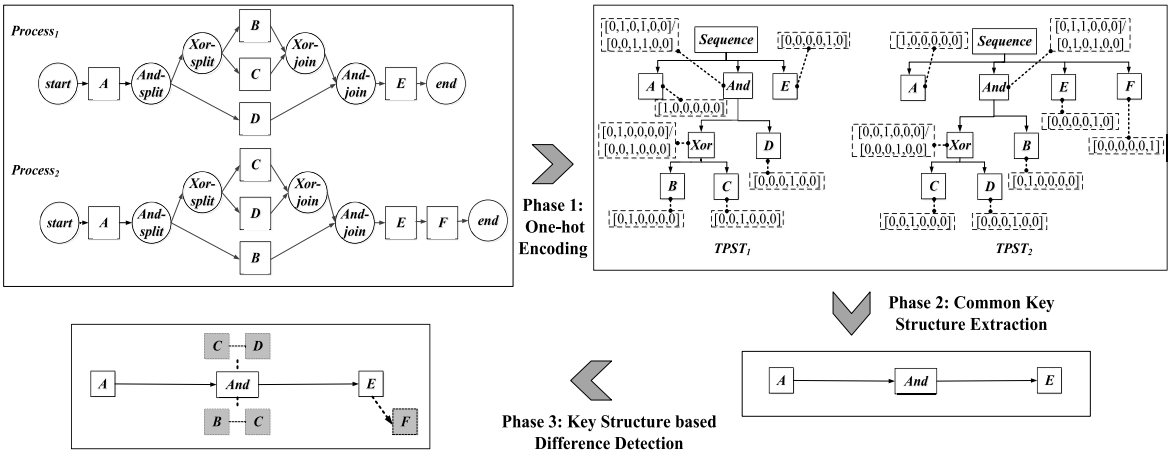


FIGURE 4. The framework of process difference detection.

task nodes from two TPSTs, and allocate a one-hot code to each node in this task union (Line 2 - 4).

As for the TPST non-leaf nodes, we design a function **setOneHot** to do the encoding for them. To achieve this goal, we iterate and encode every non-leaf node in each level from bottom to top. We start iterating from the last second level, this is because the non-leaf nodes do not appear in the last level (Line 7 - 8). Since there are four types of non-leaf nodes, i.e., *Sequence*, *Loop*, *Xor* and *And*, we divide them into three groups to encode the current iterated non-leaf node. The first group is *Xor* node, we just put all of its child's feature vectors together as its feature vector (Line 12 - 15), and the relation among these child's feature vectors is exclusive. The reason behind is that an *Xor* node will select one branch to execute when it contains more than one branches, so any branch in the corresponding conditional pattern has a certain possibility to be executed. The second group is *Sequence* and *Loop* nodes, we concatenate their child nodes' feature vectors as their feature vectors (Line 16 - 18). The reason for this is that all nodes in a sequential pattern will be executed in order, and the concatenation of feature vectors shows this sequential execution. The third group is *And* node, we perform logical "OR" operation among all of its child's feature vectors. Note that we need to insert 0 in the back of a feature vector if the lengths of the child's feature vectors are different (Line 19 - 21). The reason behind is that all nodes in a parallel pattern can be executed simultaneously, and the logical "OR" can reserve all nodes in this pattern.

Example: As shown in Figure 4, $TPST_1$ and $TPST_2$ are two TPSTs transformed from $Process_1$ and $Process_2$ in Figure 2. The task node sets of $TPST_1$ and $TPST_2$ are $\{A,B,C,D,E\}$ and $\{A,B,C,D,E,F\}$, respectively. Their task union is $\{A,B,C,D,E,F\}$, so their one-hot codes, i.e., feature vectors, are [1,0,0,0,0,0], [0,1,0,0,0,0], [0,0,1,0,0,0], [0,0,0,1,0,0], [0,0,0,0,1,0] and [0,0,0,0,0,1], respectively. The non-leaf nodes of $TPST_1$ are *Xor*, *And* and *Sequence*. We first iterate the third level of $TPST_1$ and process the non-leaf node *Xor*. Its feature vector is "[0,1,0,0,0,0]/[0,0,1,0,0,0]" since it contains two exclusively executed child nodes task *B* and *C* with one-hot vectors [0,1,0,0,0,0] and [0,0,1,0,0,0], respectively. Then, the second level is iterated and we calculate the feature vector of *And*, i.e., "[0,1,0,1,0,0]/[0,0,1,1,0,0]". It is obtained by performing logical "OR" between [0,1,0,0,0,0] and [0,0,0,1,0,0], [0,0,1,0,0,0] and [0,0,0,1,0,0], where [0,1,0,0,0,0] and [0,0,1,0,0,0] are the elements from the feature vector of *Xor* and [0,0,0,1,0,0] is task *D*'s one-hot vector. The feature vector of *Sequence* in the first level is "[1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,0] / [1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0]", which is generated by concatenating the feature vectors of its child nodes, i.e., task *A*, non-leaf node *Xor* and task *E*.

B. PHASE 2: COMMON KEY STRUCTURE EXTRACTION

The inputs of this phase are two encoded TPSTs, and the output is their common key structure.

Main Idea: The key structure of a process model abstracts its main control flow patterns executed in sequential order. A control flow pattern of a process model is represented by a non-leaf node in its corresponding TPST. To detect the common key structure of two process models, we compare the feature vectors of each possible pair of TPST nodes with the same type and determine the mapped ones. In this way, the mapped nodes and the execution relations among them form the common key structure.

Algorithm: Algorithm 2 shows how we extract the common key structure from two process models. First, we extract the nodes in the second level of a TPST if its root node is *Sequence* or *Loop*, otherwise the nodes in the first level are regarded as the candidate nodes to create the common key structure (Line 1 - 3). Then, each pair of nodes with the same type in these two *candidate* nodes are compared to decide whether they can be mapped, which includes two cases. The first case is TPST leaf node, two leaf nodes can be mapped if and only if their one-hot codes are identical (Line 6 - 9).

Algorithm 1 One-Hot Encoding

Input : Two process models p_1 and p_2
Output: Two encoded TPSTs: $tpst_1$ and $tpst_2$

1 $tpst_1$, $tpst_2$ ← transform p_1 and p_2 to TPSTs;
2 $task\_union$ ← $tpst_1$.task ∪ $tpst_2$.task;
3 **for** each task node t in $task\_union$ **do**
4 set t a one-hot code as its feature vector;
5 **return** **setOneHot**($tpst_1$) and **setOneHot**($tpst_2$);
6 **Function** setOneHot *(*TPST *t)*
7 **for** i from t.depth-2 to 1 **do**
8 $level$ ← get the nodes in the i-th level of t;
9 **for** each node cur in $level$ **do**
10 cs ← get the child nodes of cur;
11 fv ← initialize a feature vector;
12 **if** cur.type == Xor **then**
13 **for** each child c_1 in cs **do**
14 **for** each element e_1 *in* c_1*'s feature vector* **do**
15 add e_1 to fv;
16 **else if** cur.type == Sequence || Loop **then**
17 **for** each child node c_2 in cs **do**
18 fv ← fv + c_2's feature vector;
19 **else if** cur.type == And **then**
20 **for** each child node c_3 in cs **do**
21 fv ← perform the OR operation among all feature vectors of c_3;
22 set fv as cur's feature vector;
23 **return** t;

Algorithm 2 Common Key Structure Extraction

Input : Two encoded TPSTs: t_1 and t_2
Output: Common key structure: cks

1 cks ← initialize a list;
2 n_1, n_2, $mapped\_nodes$ ← initialize two lists and one map;
3 n_1 ← t_1.root.type == Sequence or Loop? Get t_1's nodes in level 1: Get t_1's nodes in level 0;
4 n_2 ← t_2.root.type == Sequence or Loop? Get t_2's nodes in level 1: Get t_2's nodes in level 0;
5 **for** each node nn_1 in n_1 **do**
6 **for** each node nn_2 *in* n_2 **do**
7 **if** nn_1.type == nn_2.type **then**
8 **if** both nn_1 *and* nn_2 are leaf nodes and nn_1*'s one-hot code* == nn_2*'s one-hot code* **then**
9 put < nn_1,nn_2 > to $mapped\_nodes$;
10 **else if** both nn_1 *and* nn_2 are non-leaf nodes **then**
11 **for** each element o_1 in nn_1*'s* feature vector **do**
12 **for** each element o_2 in nn_2.feature vector **do**
13 **if** o_1 == o_2 **then**
14 put < nn_1,nn_2 > to $mapped\_nodes$;
15 **for** each k in n_1 **do**
16 **if** $mapped\_nodes$ contains k **then**
17 add k to cks;
18 **return** cks;

The second case is non-leaf node, two non-leaf nodes can be mapped once one or more elements in their feature vectors are identical (Line 10 - 14). Finally, the common key structure is built up based on these mapped nodes, where the relationship among these mapped nodes is sequential (Line 14 - 17).

Example: As shown in Figure 4, we extract the nodes in the second level of $TPST_1$ and $TPST_2$ as the candidate nodes to construct their common key structure, since both of their root nodes are *Sequence* nodes. The key structures of $TPST_1$ and $TPST_2$ are "$A \rightarrow And \rightarrow E$" and "$A \rightarrow And \rightarrow E \rightarrow F$", respectively. For task A and E in $TPST_1$, their mapped nodes in $TPST_2$ are task A and E, respectively. Two *And* gateway nodes are also mapped, it is because one element in their one-hot vectors is identical, i.e., "[0,1,0,1,0,0]". Thus, the common key structure of $TPST_1$ and $TPST_2$ is "$A \rightarrow And \rightarrow E$".

C. PHASE 3: DIFFERENCE DETECTION

The inputs of phase 3 are two process models' common structure and their *candidate* node sets, and the output is their structural and behavioral differences.

Main Idea: Based on the common key structure, both structural and behavioral differences of two process models can be detected. For each node in the common key structure, we can determine the mapped nodes from two TPSTs, and the nodes that do not belong to the common key structure are unmapped nodes. These unmapped nodes are the differences of two process models in terms of topology structure, where the unmapped nodes show that they just exist in one process model. A mapped pair of non-leaf nodes may contain the same structure but different behaviors, that is, some elements of their feature vectors are identical while the rest elements are different. These different elements are used to describe the behavioral difference.

Algorithm: Algorithm 3 calculates both structural and behavioral differences between two process models by the following steps. First, since the differences are described by node labels, we split the *candidate* nodes' labels of two TPSTs by the node labels of common key structure and get the structural difference, i.e., the unmapped nodes' labels (Line 1 - 4). Then, for every mapped pair of non-leaf nodes,

Algorithm 3 Difference Detection

Input : Common key structure *cks*, two TPSTs' candidate node sets n_1 and n_2 and their mapped node set *mapped_nodes*

Output: The structural differences *s_diff* and the behavioral differences *b_diff*

```
1  cks_str ← the node label set of cks;
2  n1_str, n2_str ← the node labels of n1 and n2;
3  add n1_str.split("["+cks_str+"]") to s_diff;
4  add n2_str.split("["+cks_str+"]") to s_diff;
5  map ← key is the position of 1 and value is the
   corresponding task node label in all one-hot codes;
6  b_diff ← initialize a map;
7  for each < k,v > in mapped_nodes do
8      b_diff 1, b_diff 2 ← initialize two lists;
9      if both k and v are non-leaf nodes and k or v contains
       more than one elements then
10         o1 ← get all elements of k's feature vector;
11         o2 ← get all elements of v's feature vector;
12         ones ← get the same elements of k and v;
13         if ones.size ≠ o1.size then
14             o1 ← o1 − ones;
15         if ones.size ≠ o2.size then
16             o2 ← o2 − ones;
17         for each oo1 in o1 do
18             for i from 0 to oo1.length do
19                 add map.get(i % map.size) to b_diff 1
20         for each oo2 in o2 do
21             for j from 0 to oo2.length do
22                 add map.get(j % map.size) to b_diff 2
23     add <k, [b_diff 1, b_diff 2]> to b_diff;
24 return s_diff and b_diff;
```

we get their corresponding feature vectors and extract their identical and different elements (Line 7 - 12). In this way, the behavioral differences can be described by these different elements, which is implemented by a map where the key is the position of 1 in the one-hot vector of the task nodes and value is the corresponding task node's label. According to this map, we can find the task nodes as well as their execution orders, i.e., different execution paths between two mapped control flow patterns (Line 13 - 22).

Example: As shown in Figure 4, the common key structure of two process models is "$A \rightarrow And \rightarrow E$", and their *candidate* nodes are "{*A*, *And*, *E*}" and "{*A*, *And*, *E*, *F*}", respectively. The mapped nodes are *A* and *A*, *And* and *And*, *E* and *E*, so task *F* in $Process_2$ is the unmapped node, which causes the structural difference. For *And* node, the feature vectors of its corresponding two mapped *And* nodes are "[0,1,0,1,0,0]/[0,0,1,1,0,0]" and "[0,1,1,0,0,0]/[0,1,0,1,0,0]", respectively. Their different elements are "[0,0,1,1,0,0]" and "[0,1,1,0,0,0]". Since the map that records the position of 1 in the one-hot vector and its corresponding node label is "{0:*A*, 1:*B*, 2:*C*, 3:*D*, 4:*E*, 5:*F*}", we get the different paths in terms of *And* are "$C \rightarrow D$" and "$B \rightarrow C$", respectively.

IV. EXPERIMENTAL EVALUATION

This section provides experimental evaluation of the proposed approach. We first present a case study to analyze the differences between two process models in the real world, which comes from the projects conducted in the health care domain [23]. Then, we investigate the execution time based on a mixture of real and synthetic data sets. The real part is from an existing dataset of IBM [24]. We make some modifications on the real models to obtain 24 synthetic process models that are divided into 4 groups, where each group includes one *baseline* model that is used to be compared with other models in this group [25]. The *baseline* model includes 62 task nodes and 12 patterns. For the rest 20 models, the first three groups separately contain conditional, parallel and loop patterns, and the pattern numbers are 4, 8, 12, 16, 20 separately in each group. The last group consists of different patterns, and the numbers of task are 20, 40, 60, 80 and 100, respectively. All experiments were evaluated on a machine with Intel(R) Core(TM) i7-6650U CPU @ 2.20GHz 2.21GHz, running JDK 1.8 and Windows 10.

A. CASE STUDY

As demonstrated in Figure 5, there are two process models *Process* 1 and *Process* 2 to perform the examination in a hospital. To perform an examination, the patient first selects the related examination and then performs it. Finally, the patient will obtain his medical report. The rectangle nodes in two process models with the same label can perform the same task, i.e., they are equivalent.

The differences between *Process* 1 and *Process* 2 are shown in Figure 6. Their common key structure consists of four parts, i.e., selecting examination (task *A*), the conditional pattern that consists of preparing and transporting patient (task *I* and *N*), performing examination (task *J*) and the parallel pattern that includes transporting patient back (task *O*) and creating a medical report (task *L*). That is, the tasks in these four parts are the common tasks to be executed in both *Process* 1 and *Process* 2. Besides, the execution orders among them are identical, i.e., the patient will first select an examination, then he will be prepared or transported, next is to perform the examination, and finally creating medical report and transporting him back.

For the structural differences, there is an extra task *E* in *Process* 1, i.e., registering the patient, between task *A* and the conditional pattern *Xor*. That is, the patient needs to be registered before being prepared or transported. While in *Process* 2, task *B*, i.e., second opinion by other doctor, exists in the same position as task *E*. Besides, some additional tasks are executed in *Process* 1 after performing the examination (task *J*) and before the parallel pattern that consists of

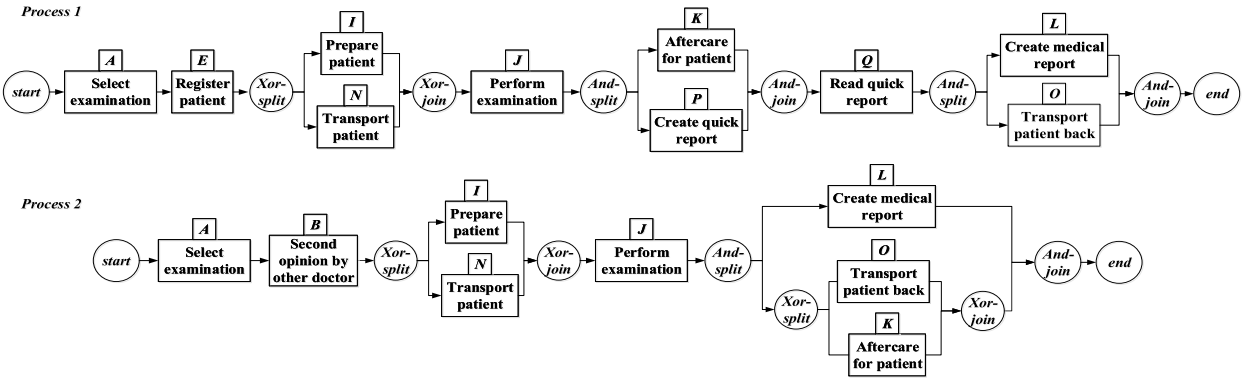


FIGURE 5. A case study.

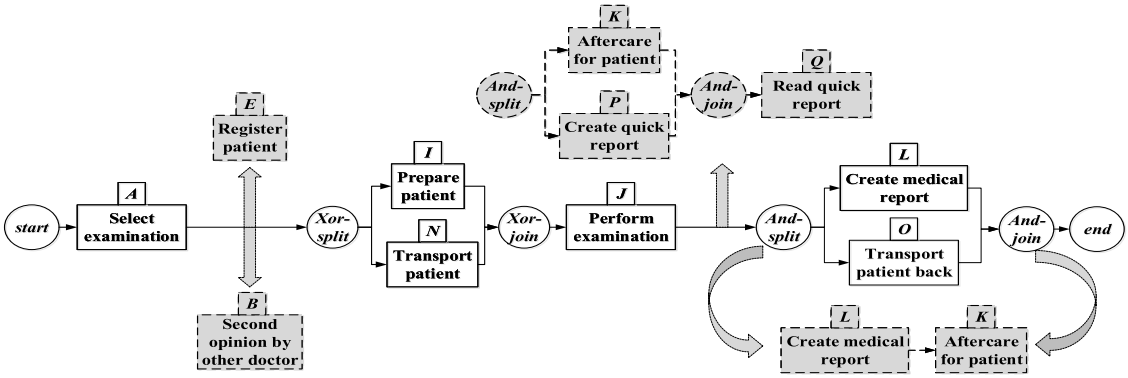
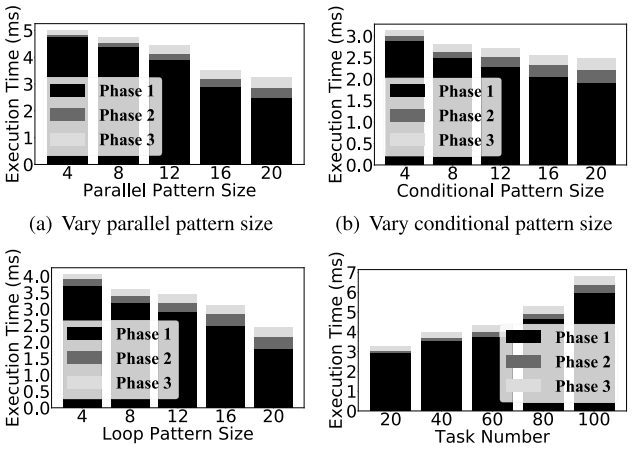


FIGURE 6. The key structure based difference of case study.

task L and O. These additional tasks are after-caring for the patient and creating a quick report (task K and P) that form a parallel pattern, and reading quick report (task Q). *Process* 2 omits these tasks, and directly creates medical report and transports patient back after performing the examination.

For the behavioral differences, the common execution path of the last parallel pattern in two process models is $L \rightarrow O$ (or $O \rightarrow L$), while an extra execution path exists in *Process* 2, i.e., $L \rightarrow K$ (or $K \rightarrow L$). That is, creating medical report and transporting patient back can be concurrently executed in both process models. While in *Process* 2, apart from these two tasks, creating medical report (task L) and after-caring for patient (task K) can also be concurrently executed. Thus, *Process* 2 will randomly select task O or task K to be concurrently executed with task L.



(a) Vary parallel pattern size (b) Vary conditional pattern size

(c) Vary loop pattern size (d) Vary task number

FIGURE 7. Efficiency study.

B. EFFICIENCY STUDY

In this section, we study the internals of the proposed approach, i.e., three phases of the proposed approach: *one-hot encoding*, *common key structure extraction*, and *difference detection*. Figure 7 shows the ratio of the execution time spent for each phase.

In Figure 7 (a), (b) and (c), we use the first three groups of process models in the data set to investigate the execution time by varying the pattern size from 4 to 20 while fixing the pattern type to parallel, conditional and loop, respectively. We can see that the total execution time for these three pattern types decreases as the pattern size increases. This is because the total task number of all process models is fixed to 62, the complexity of the pattern decreases with the pattern number growing bigger, while our proposed approach is more sensitive to the pattern complexity than the pattern size. Besides, Phase 1 spends almost 20 orders of magnitude more

than Phase 2 and Phase 3, and the execution time of Phase 1 also decreases with the pattern size increasing. The reason behind is that there are 62 task nodes and less than 20 control flow patterns in each process model to be processed in Phase 1, while only the control flow patterns are processed in Phase 2 and Phase 3. What's more, the more simple the pattern, the less execution time Phase 1 spends in encoding the TPST non-leaf node.

In Figure 7 (d), we fix the pattern size to 10 and vary the task number to 20, 40, 60, 80 and 100, respectively. We can observe that the task number has a relatively apparent impact on the total execution time, i.e., as the task number increases, the total execution time grows. This is because the pattern complexity increases with the task number getting bigger, each phase will spend more time in processing these patterns. Besides, Phase 1 spends the most execution time compared with the other two phases, and the task number has the most obvious impact on Phase 1. The reason behind this is that Phase 1 needs to process both task nodes and control flow patterns, while Phase 2 and 3 are less sensitive to the task node number compared with Phase 1. Thus, the more task nodes, the more time Phase 1 spends in processing them.

V. RELATED WORK

Detecting difference between two process models has been wildly studied in recent years, which can be classified into three categories: (1) Edit script based process difference detection, where one process model can be transformed into another by this edit script, (2) behavior based process difference detection, and (3) process similarity calculation.

A. EDIT SCRIPT BASED PROCESS DIFFERENCE DETECTION

Fan *et al.* [5] detected the difference between two process models by parsing them into task-based process structure trees (TPSTs), and the edit scripts in terms of fragments and nodes are generated. Cao *et al.* [4] also transformed the process models to their corresponding process structure trees, and the subtree deletion and subtree insertion operations were generated based on their largest common sub-tree. Küster *et al.* [26] detected and resolved the differences between two process models based on the decomposition of process models in the absence of a change log. Li *et al.* [27] used digital logic to evaluate the similarity and difference between two process models based on high-level change operations. In our previous work [28], the differences between two process models were detected by splitting the process models into fragments and the fragments were represented by feature vectors, whether two fragments were identical or not was decided by comparing their corresponding feature vectors. However, only focusing on the structural feature cannot reveal different behaviors since the identical structure of two process models may contain different behaviors.

B. BEHAVIOR BASED PROCESS DIFFERENCE DETECTION

Ballambettu *et al.* [29] identified the reasons for the key differences in terms of control-flow and performance among the process variants. Bolt *et al.* [30] detected the behavioral differences between process variants in terms of process metric, e.g., performance, based on annotated transition systems. Armas-Cervantes *et al.* [7], [8] detected and diagnosed behavioral differences between business process models based on a translation from the process model into asymmetric event structure. Yan *et al.* [31] detected behavioral differences between two process models by comparing dependency and trace sets of features in process models. Cao *et al.* [32] detected the difference between two process models based on their basis paths. Jovanovikj *et al.* [33] proposed an approach for detecting, visualizing, and resolving dataflow differences of business process models. However, the same behavior of two process models does not mean their structures are also identical.

Actually, structure and behavior are two important features of a process model, However, all the mentioned methods just focus on one feature, i.e., structure or behavior, which leads to the loss of information on other process features. To provide more information about process difference, our method takes both structural and behavioral features into consideration.

C. PROCESS SIMILARITY CALCULATION

Huang *et al.* [34] proposed an improved two-stage exact query approach based on graph structure similarity. Assy *et al.* [35] measured the process similarity based on the contextual similarity, where the similarity of the context surrounding activities was considered. The global contextual similarity between process activities was computed by similarity resonance. Liu *et al.* [36] proposed an approach to measure the business process behavior similarity based on the so-called Extended Transition Relation set, which was an extended transition relation set containing direct causal transition relations, minimum concurrent transition relations and transitive causal transition relations. Zhou *et al.* [37] measured the business process similarity by considering both process models and process logs, where the process models were pre-defined descriptions of business processes, and the process logs was regarded as an objective observation of the actual process execution behavior. However, the similarity score of two process models can only measure their difference degree, while cannot reflect the composition of the difference.

VI. CONCLUSION

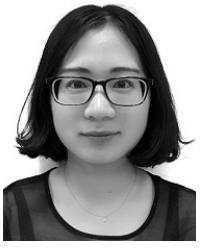
In this paper, a process difference detection framework based on edge network is used to improve the computational efficiency, where the edges can detect differences between two process models. To detect the differences between two process models, we propose a one-hot encoding based approach to calculate the differences between two process models. The common key structure of two process models is first determined, and then both the structural and behavioral differences can be displayed beyond this common key structure. The case study and the efficiency study show that the proposed approach can be applied in real life.

The limitation of our approach is that one-hot encoding is discrete and sparse, and the one-hot vector of a node

becomes large when there exists a large number of nodes in a process model, which leads to curse of dimensionality. Besides, the computational complexity is quickly increasing with the increment of one-hot vector, which results in spending a lot of time in encoding the nodes. In the future, we are going to solve this limitation. Besides, we are going to take more features of a process into consideration to detect the difference between two process models, such as resource and resource constraints. What's more, we will conduct more extensive experiments against the state of the art.

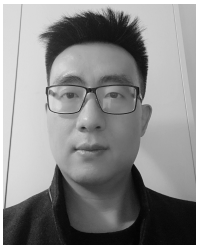
REFERENCES

[1] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "An introduction to model versioning," in *Formal Methods for Model-Driven Engineering*. Bertinoro, Italy: Springer-Verlag, 2012, pp. 336–398.
[2] N. Macedo, T. Jorge, and A. Cunha, "A feature-based classification of model repair approaches," *IEEE Trans. Softw. Eng.*, vol. 43, no. 7, pp. 615–640, Jul. 2016.
[3] H. Gao, Y. Duan, H. Miao, and Y. Yin, "An approach to data consistency checking for the dynamic replacement of service process," *IEEE Access*, vol. 5, pp. 11700–11711, 2017.
[4] J. Cao, Y. Yao, and Y. Wang, "Mining change operations for workflow platform as a service," in *World Wide Web*, vol. 18, no. 4, pp. 1071–1092, Jul. 2015.
[5] J. Fan, J. Wang, W. An, B. Cao, and T. Dong, "Detecting difference between process models based on the refined process structure tree," *Mobile Inf. Syst.*, vol. 2017, Mar. 2017, Art. no. 6389567.
[6] J.-R. Rehse, P. Fettke, and P. Loos, "A graph-theoretic method for the inductive development of reference process models," *Softw. Syst. Model.*, vol. 16, no. 3, pp. 833–873, 2017.
[7] A. Armas-Cervantes, P. Baldan, M. Dumas, and L. García-Bañuelos, "Behavioral comparison of process models based on canonically reduced event structures," in *Business Process Management*. Haifa, Israel: Springer, 2014, pp. 267–282.
[8] A. Armas-Cervantes, P. Baldan, M. Dumas, and L. García-Bañuelos, "Diagnosing behavioral differences between business process models," *Inf. Syst.*, vol. 56, pp. 304–325, Mar. 2016.
[9] A. Bolt, M. de Leoni, and W. M. P. van der Aalst, "Process variant comparison: Using event logs to detect differences in behavior and business rules," *Inf. Syst.*, vol. 74, pp. 53–66, May 2018.
[10] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "QoS prediction for service recommendation with deep feature learning in edge computing environment," *Mobile Netw. Appl.*, pp. 1–11, 2019.
[11] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[12] H. Gao, W. Huang, Y. Duan, X. Yang, and Q. Zou, "Research on cost-driven services composition in an uncertain environment," *J. Internet Technol.*, vol. 20, no. 3, pp. 755–769, 2019.
[13] Y. Yin, W. Xu, Y. Xu, H. Li, and L. Yu, "Collaborative QoS prediction for mobile service with data filtering and SlopeOne model," *Mobile Inf. Syst.*, vol. 2017, Jun. 2017, Art. no. 7356213.
[14] J. Xu, C. Liu, X. Zhao, S. Yongchareon, and Z. Ding, "Resource management for business process scheduling in the presence of availability constraints," *ACM Trans. Manage. Inf. Syst.*, vol. 7, no. 3, Oct. 2016, Art. no. 9.
[15] M. Fu, C. M. Wong, H. Zhu, Y. Huang, Y. Li, X. Zheng, J. Wu, J. Yang, and C.-M. Vong, "Dalim: Machine learning based intelligent lucky money determination for large-scale e-commerce businesses," in *Service-Oriented Computing*. Hangzhou, China: Springer, 2018, pp. 740–755.
[16] H. Gao, K. Zhang, J. Yang, F. Wu, and H. Liu, "Applying improved particle swarm optimization for dynamic service composition focusing on quality of service evaluations under hybrid networks," *Int. J. Distrib. Sensor Netw.*, vol. 14, no. 2, Feb. 2018, Art. no. 1550147718761583.
[17] L. Qi, Y. Chen, Y. Yuan, S. Fu, X. Zhang, and X. Xu, "A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems," in *World Wide Web*. Taipei, Taiwan, 2019, pp. 1–23.
[18] Y. Yin, S. Aihua, G. Min, X. Yueshen, and W. Shuoping, "QoS prediction for Web service recommendation with network location-aware neighbor selection," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 4, pp. 611–632, 2016.
[19] M. E. Acevedo-Mosqueda, C. Yáñez-Márquez, and I. López-Yáñez, "Alpha-beta bidirectional associative memories based translator," *Comput. Sci. Netw. Secur.*, vol. 6, no. 5A, pp. 190–194, 2006.
[20] H. Zheng, J. Yang, and W. Zhao, "Probabilistic QoS aggregations for service composition," *ACM Trans. Web*, vol. 10, no. 2, 2016, Art. no. 12.
[21] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.
[22] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," *Data Knowl. Eng.*, vol. 68, no. 9, pp. 793–818, Sep. 2009.
[23] C. Li, M. Reichert, and A. Wombacher, "The minadept clustering approach for discovering reference process models out of process variants," *Cooperat. Inf. Syst.*, vol. 19, nos. 3–4, pp. 159–203, 2010.
[24] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf, "Instantaneous soundness checking of industrial business process models," in *BPM: Business Process Management*. Berlin, Germany: Springer, 2009, pp. 278–293.
[25] J. Wang, B. Cao, W. An, J. Fan, and J. Yin, "A benchmark dataset for evaluating process similarity search methods," in *Proc. ICWS*, Jun. 2017, pp. 914–917.
[26] J. M. Küster, C. Gerth, A. Förster, and G. Engels, "Detecting and resolving process model differences in the absence of a change log," in *BPM: Business Process Management*. Milan, Italy: Springer, 2008, pp. 244–260.
[27] C. Li, M. Reichert, and A. Wombacher, "On measuring process model similarity based on high-level change operations," in *Proc. Int. Conf. Conceptual Modeling*. Barcelona, Spain: Springer, 2008, pp. 248–264.
[28] J. Wang, B. Cao, J. Fan, and T. Dong, "FB-diff: A feature based difference detection algorithm for process models," in *Proc. ICWS*, Jun. 2017, pp. 604–611.
[29] N. P. Ballambettu, M. A. Suresh, and R. J. C. Bose, "Analyzing process variants to understand differences in key performance indices," in *Advanced Information Systems Engineering*. Essen, Germany: Springer, 2017, pp. 298–313.
[30] A. Bolt, M. de Leoni, and W. M. van der Aalst, "A visual approach to spot statistically-significant differences in event logs based on process metrics," in *Advanced Information Systems Engineering*. Ljubljana, Slovenia: Springer, 2016, pp. 151–166.
[31] Z. Yan, Y. Wang, L. Wen, and J. Wang, "Efficient behavioral-difference detection between business process models," in *Proc. OTM Confederated Int. Conf. 'Move Meaningful Internet Syst.'* Amantea, Italy: Springer, 2014, pp. 220–236.
[32] B. Cao, F. Hong, J. Wang, J. Fan, and M. Lv, "Workflow difference detection based on basis paths," *Eng. Appl. Artif. Intell.*, vol. 81, pp. 420–427, May 2019.
[33] I. Jovanovikj, E. Yigitbas, C. Gerth, S. Sauer, and G. Engels, "Detection and resolution of data-flow differences in business process models," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.* Springer, 2019, pp. 145–157.
[34] H. Huang, R. Peng, and Z. Feng, "Efficient and exact query of large process model repositories in cloud workflow systems," *IEEE Trans. Services Comput.*, vol. 11, no. 5, pp. 821–832, Sep./Oct. 2015.
[35] N. Assy, B. F. van Dongen, and W. M. van der Aalst, "Similarity resonance for improving process model matching accuracy," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, 2018, pp. 86–93.
[36] C. Liu, Q. Zeng, H. Duan, S. Gao, and C. Zhou, "Towards comprehensive support for business process behavior similarity measure," *Trans. Inf. Syst.*, vol. 102, no. 3, pp. 588–597, 2018.
[37] C. Zhou, C. Liu, Q. Zeng, Z. Lin, and H. Duan, "A comprehensive process similarity measure based on models and logs," *IEEE Access*, vol. 7, pp. 69257–69273, 2019.

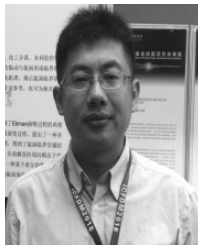


JIAXING WANG was born in Jiaxing, China, in 1990. She received the B.E. degree in software engineering and the Ph.D. degree in control science and engineering from the Zhejiang University of Technology, Hangzhou, China, in 2013 and 2019, respectively, where she is currently a Postdoctoral with the College of Computer Science and Technology. Her research interest includes business process management.

BIN CAO received the Ph.D. degree in computer science from Zhejiang University, China, in 2013. He then worked as a Research Associate with The Hong Kong University of Science and Technology, and Noah's Ark Lab, Huawei. He joined the Zhejiang University of Technology, Hangzhou, China, in 2014, and is currently an Associate Professor with the College of Computer Science. His research interests include business process management and data mining.



XI ZHENG received the bachelor's degree in computer information system from FuDan, the master's degree in computer and information science from UNSW, and the Ph.D. degree in software engineering from UT Austin. He served as a Chief Solution Architect for Menulog, Australia; currently an Assistant Professor/Lecturer in software engineering with Macquarie University. He is also specialized in service computing, the IoT security, and reliability analysis. He published more than 40 high-quality publications in top journals and conferences, such as PerCOM, ICSE, ICCPS, the IEEE Systems Journal, and the *ACM Transactions on Embedded Computing Systems*. He was awarded Deakin Research Outstanding Award, in 2016. He was awarded the best paper in Australian distributed computing and doctoral conference, in 2017. He was a Reviewer for top journals and conferences, such as the IEEE Systems Journal, *ACM Transactions on Design Automation of Electronic Systems*, *Pervasive and Mobile Computing*, the IEEE Transaction on Cloud Computing, and PerCOM.



DAPENG TAN received the Ph.D. degree from the College of Mechanical and Energy Engineering, Zhejiang University, China, in 2008. Since 2008, he has been with the Key Laboratory of E&M, Ministry of Education and Zhejiang Province, Zhejiang University of Technology, as a Lecturer, and became an Associate Professor, in 2010. He is currently a Professor with the College of Mechanical and Energy Engineering. He performed research in the areas of embedded system technology, advanced manufacturing technology, and metallurgy process automatic control. His research interests include real-time intelligent networks and industrial process monitoring and diagnosis.

JING FAN received the B.S., M.S., and Ph.D. degrees in computer science from Zhejiang University, China, in 1990, 1993, and 2003, respectively. She is currently a Professor with the School of Computer Science and Technology, and the Director of the Institute of Computer Software, Zhejiang University of Technology, China. Her current research interests include service computing and software middleware. She is also the Director of China Computer Federation (CCF), and a member of CCF Technical Committee on Service Computing.