Future Generation Computer Systems 89 (2018) 670–684

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A fog computing based concept drift adaptive process mining framework for mobile APPs

Tao Huang [a], Boyi Xu [b], Hongming Cai [a,*], Jiawei Du [a], Kuo-Ming Chao [c], Chengxi Huang [a]

[a] School of Software, Shanghai Jiao Tong University, 800 Dongchuan Rd, Shanghai, China
[b] Antai College of Economics & Management, Shanghai Jiao Tong University, 1954 Huashan RD, Shanghai, China
[c] Faculty of Engineering, Environment & Computing, Coventry University, Coventry, UK

HIGHLIGHTS

- Context aware method to efficiently reflect business process by distributed log.
- Fog computing based framework to analyze logs more precisely and promptly.
- Concept drift adaptive algorithm with consideration of active drift detections.
- Cloud service governance method for scenarios of APPs continuously evolvement.

ARTICLE INFO

Article history:
Received 31 December 2017
Received in revised form 15 April 2018
Accepted 17 July 2018
Available online 23 July 2018

Keywords:
Process mining
Concept drift
Fog computing
Log analysis
Cloud governance

ABSTRACT

Mobile applications are widely used to provide users convenient and friendly service experiences. Meanwhile, service logs generated by mobile applications are analyzed to obtain user behavior patterns for monitoring and optimizing mobile application performances. However, due to the frequent updates in mobile application, situations of concept drifts often occur in service log streams, which lead to challenges in mobile process mining. In this paper, a novel framework is proposed to solve the above problems by combining fog-computing-based concept drift detecting with cloud-computing-based process mining. Firstly, incomplete log data are preprocessed using fog-computing technologies to provide more accurate log contexts and lower overhead. Then, concept drift detecting methods are used in cloud computing layer to deal with the transfer of mobile applications from one version to another. Finally, experimental results demonstrate that our framework can deduce missed case identifiers for logs when concept drifts happen.

1. Introduction

Nowadays, with the rapid development of mobile Internet, many Internet companies have developed various mobile APPs (Applications) based on smartphones and other mobile devices to provide convenient services to clients, which have greatly changed our lives. The patterns of users' behaviors are demonstrated when they manipulate mobile APPs. These kinds of user behavior models are very important in some activities such as APP UI enhancements and system monitoring, at design time and runtime respectively, which require the up-to-date user behavior models as foundations for real-time assessment and further analysis.

Mobile applications generally generate large amounts of service logs that could be further analyzed. Process mining is one of the useful techniques to discover user behavior models through these log data [1], including process model discovery [2], conformance checking [3] and enhancements [4].

However, for most mobile APPs, there are some problems existing in process mining. Firstly, because most mobile APPs are not Process-Aware Information Systems (PAISs) [1], log data often lack case IDs to tag process instances for process mining. So case IDs need to be added to log data before they are analyzed.

Secondly, it is necessary to preprocess log data locally. Different process instances may occur in the same log file/stream, for example, in m-health APPs, services of making appointments and querying clinic reports belong to different business processes. If these services are required concurrently to the Web servers of m-health APPs, log data may be recorded in the same log file/stream and need to be split in preprocessing. There has been some studies on processing data from mobile devices. [5] provides a functional framework that identifies the acquisition, management, processing and mining areas of IoT big data. [6] and [7] propose mobility-aware hierarchical architecture for data collection in local and processing in cloud, but both lack a local-centralized structure for

* Corresponding author.
E-mail address: hmcai@sjtu.edu.cn (H. Cai).

https://doi.org/10.1016/j.future.2018.07.034
0167-739X/

data preprocessing. [8] regards smartphone as a mobile gateway to provide an interface between different mobile devices and the Internet, however, due to the completely localized architecture, [8] cannot perform high-performance computing on large amounts of data. Therefore, a novel log processing framework is needed.

Thirdly, due to the frequent releases of mobile APPs, the concepts that are used to depict users' behavioral patterns may change with the update of APPs versions. This is called concept drift. In order to effectively mine process logs, concept drifts should be detected automatically with the changes of APPs. Therefore, to address the above issues, in this paper, a concept drift adaptive framework of process mining is proposed using fog computing based architecture to balance network workload and offer comprehensively analyzed results. Contributions of this paper are as follows:

- A context aware method is presented to merge distributed service logs into event logs to more efficiently reflect the business process.
- A fog computing based log processing framework is proposed to analyze logs more precisely and promptly.
- A concept drift adaptive process discovery algorithm is designed to discover process models from event stream with the consideration of active concept drift detections.
- A cloud service governance method is discussed to deal with the application scenarios of APPs continuously evolvement.

The rest of this paper is structured as follows. Section 2 presents the related works and Section 3 gives the overview of our framework. Then Section 4 explains the details of our approach on the general idea of log collection, data disposing, process model discovery, and cloud service governance. In Section 5, the experimentation and a case study on a Chinese m-health APP are discussed. Section 6 makes some comparisons between our methods and others. Section 7 concludes the paper.

2. Related work

Related work can be categorized into four parts, including state-of-the-art process discovery algorithms to find models from static event logs as well as event streams, concept drift detection approaches to figure out the changes of process models, applications of process mining in real life and development of fog computing since presented.

(1) Process discovery. Some state-of-the-art methods in process discovery from multiple perspectives have been proposed. In [9], the well-known α algorithm is defined to find the control-flow model in form of Petri Net. Some extensions about α algorithm [10,11] are reported to have successfully overcome the defeats of the original α algorithm. Heuristics Miner Algorithm is presented in [12] to handle with the noise in event logs. For online process discovery from event streams, several algorithms are discussed to discover the process model in a single pass of streaming data. Burattin et al. in [13] proposes a Sliding Window (SW), Lossy Counting (LC) and Lossy Counting with Budget (LCB) based algorithm to adapt Heuristic Miner Algorithm to event streams. These ideas are also adopted in [14] which focuses on Declarative Model discovery from event streams. [15] does another modification to Heuristic Miner Algorithm by building prefix-trees to extract sequential patterns from event streams and performing decaying mechanisms, so the Heuristic Miner Algorithm can have better performance in the streaming environment. Moreover, a block-structured process is discovered based on a divide-and-conquer algorithm (Constructs Competition Miner, CCM) in [16].

(2) Concept drift detection. Concept drifts refer to the changes of process models over time. In [17], handling of concept drifts is classified into three parts, consisting of Change (Point) Detection, Change Localization and Characterization, and Unravel Process Evolution. Four categories including sudden, gradual, recurring and incremental drifts are summarized and a general framework to detect concept drifts offline is presented in [18]. For online detections, the first mechanism based on estimating abstract interpretations is demonstrated in [19]. Maggi et al. in [20] presents an online process discovery method to identify change points by evaluating constraint activations while mining. Another online detection method is proposed in [21] to figure out the drift points by continuous comparisons on the detection window and the reference window based on α^+ relations.

(3) Applications of process mining in real life. Some studies on handling unlabeled logs in real life and how to apply process mining to satisfy complex analytical requests are reported in recent years. [22] presents an Expectation Maximization based method to discover the model from unlabeled event logs and perform the Case ID completion. [23] proposes a complete search approach over the search space. Based on behavioral profiles, heuristics data and unlabeled events, [24] and [25] deduce case IDs by building case decision trees. In addition, process mining is applied to service management in recent studies since process models such as Petri Net lay the foundation of service computing [26,27]. Service choreographies are considered declarative in [28] and a declarative language DecSerFlow is proposed to model service choreographies. [29] takes use of α^+ algorithm for anomaly detection problems. And in [30], an m-health monitoring system is designed based on process mining as the healthcare data analysis layer for personalized treatment plan selections.

(4) Fog computing. Several studies on definition and application of fog computing have been done in recent years. Fog computing is regarded not as a substitute but instead as a powerful complement to the cloud computing in [31], and [32] defines fog computing as an extension of cloud computing to the edge of the network. [33] thinks fog comprehends technologies as diverse as cloud, sensor networks, peer-to-peer networks, network virtualization functions or configuration management techniques. [34] elaborates the motivation and advantages of fog computing, and analyze its applications in a series of real scenarios, such as smart grid. Mobility of fog computing is emphasized in [35] meanwhile. As for specific applications, [36] proposes a fog-driven IoT architecture for patient-centric IoT eHealth ecosystem to empower handling of complex data in terms of its variety, speed, and latency; [37] uses edge servers in fog computing architecture to improve web sites performance; and an architecture of smart gateway with fog computing is presented to preprocess and trim data before sending them to the cloud in [38]. However, there still existing some security issues, such as discussed in [34] and [39], users' data is easily stolen in fog sometimes. Moreover, [40] holistically analyzes the security threats, challenges, and mechanisms inherent in all edge paradigms, while highlighting potential synergies and venues of collaboration.

To sum up, besides process mining on static logs, recent studies start to analyze streaming data. And the majority of these discovery algorithms assume that the process remains in steady state without changing while being analyzed. Some online techniques distinguish concept drifts by conformance checks afterwards, making the mined model to sense the changes of the processes and mine the up-to-date model relatively, which is not suitable for mobile APPs that have frequent version releases. Moreover, the incompleteness of log data urges to perform case ID completions before process mining. Existing methods are not well suitable for mobile APPs, because mobile specific log data, such as spatial and temporal information, which are beneficial for more precise log analysis and less computation complexity, are not considered. Meanwhile, for mobile APPs application, it is more suitable for log

data to be gathered and preprocessed by fog computing architecture, whose location-awareness property can improve data disposing capability and reduce costs of the whole system. In addition, it will be helpful for mobile APPs' evolvement if different versions of process models could be mined from log streams. Therefore, stream process mining methods for mobile APPs are required to be studied.

3. Framework

As Fig. 1 shows, we propose a fog computing based framework for concept drift adaptive process mining. This framework is composed of four layers: log generation layer, fog service layer, cloud service layer and application layer. In log generation layer, users operate their mobile applications. By local area network, such as WIFI, logs of these operations are transmitted to the fog service layer.

The fog service layer is defined as small scale of local networks consisting of several network nodes, like routers and gateways, with limited computing ability. In our framework, the fog service layer is responsible for data disposing. If a service log item is captured by a fog service node, data disposing strategy will be carried out to preprocess service log items and generate event logs according to the *Trace Event Model*. The event log items will be converted into traces and sent to the cloud service layer through wide area network.

In the cloud service layer, some basic information of the corresponding events is extracted and stored in the *Cached Info Structure*. Considering the frequent changes in versions of mobile APPs, concept drift detections are conducted on sliding windows to evaluate the degree of the changes of process models. Once the concept drift is recognized, the current process model will be persisted as a past version, and the *Cached Info Structure* will be updated to mitigate the impact caused by concept drifts. If no drift is detected or the coping with drifts is finished, the mining algorithm will be applied to discover the process model from the log streams.

Finally, based on the multiple versions of business process models stored in the *Process Model Repository*, continuous changes of these models can be used to govern the cloud serves in the application layer.

4. Fog computing based concept drift adaptive process mining

In this section, we introduce the main methods of our paper, including log collection, fog computing based data disposing, concept drift adaptive process mining and cloud service governance.

4.1. Log collection in log generation layer

With the rapid development of smart devices and mobile internet, internet companies, like Alibaba and Tencent, have released various mobile applications cooperating with traditional industries, with which users can shop online, make an appointment with doctor online, etc.

As Fig. 2 shows, users access mobile services through wireless local area networks (LANs). Wireless LANs consist of routers or switches with the ability of wireless signal transmission, which are spatially separated from each other and serve users, such as mobile devices, to access services in different locations by wireless signals. Nodes in one wireless LAN can share messages through cables linking them together, and connect with wide area network (WAN) outside through other cables. When users access services through wireless LANs, service logs will be generated and uploaded by application programming interface (API) designed by developer in advance to the nearest node of this LAN, in fog service layer in our method, as shown in Fig. 1. Then, in the very same layer, these logs will be stored temporarily and preprocessed. Initial operations of business processes are assumed delegated in APPs in our method, therefore, if an operation is not triggered by the previous activity in the business process or delegated as initial operations, its log data should be permitted from uploading to fog nodes by the APPs.

4.2. Data disposing in fog service layer

As we have mentioned before, we expect that the input of our process-mining framework is a set of complete service log items, which contain not only service description and context data but also the case identifier. However, the case IDs are always missed in the raw service logs in most situations, especially in mobile environment. Logs uploaded by mobile applications from log generation layer need to be preprocessed in fog service layer before being used for further analysis.

4.2.1. From service log to event log based on context data

In the *Trace Event Model* defined in process mining, an event is defined as a triple containing a case identifier, an activity label and a timestamp. A trace is defined as a sequence of events that case IDs are identical. Based on the above definition, we propose a context aware method with the following two steps.

First, we extract the service name in log items as the field of *activity* of corresponding event and the request time as the field of *timestamp*, and save location information meanwhile.

Second, we observe that when mobile applications are executed, they are often expected to be finished continuously. So it is reasonable to calculate the continuity between two events to judge whether they belong to same process instance. In our method, the execution location and time of services are used to calculate the service continuity from temporal and spatial perspectives. We define two thresholds, δ_{time} and δ_{space}. If the time interval between two events is less than δ_{time} and the distance between two events is less than δ_{space}, they are considered to be continuous. The temporal and spatial continuity can help us judge whether two events should be assigned the same case ID.

4.2.2. Preprocessing in fog service layer

In the fog service layer, fog nodes deployed in different locations provide wireless-network-access services to mobile application users, which constitute the basic framework of the fog computing, as shown in Fig. 3.

In fog computing architecture, from the viewpoint of the mobile APPs users, the target of log uploading of mobile applications is no longer the remote cloud servers, instead, it is the fog nodes that provide current wireless network signals. Therefore, mobile applications need to obtain the IP addresses of the corresponding fog nodes firstly. From the viewpoint of fog nodes, on the other hand, when users initiate service requests, the corresponding fog nodes are the first nodes that receive the requests. In this kind of situation, the fog nodes may act in the following ways:

- If it is a network request except from log uploading, the fog node will act just like a router, transmitting the request to a wide area network.
- If it is a log-uploading request, the fog node will capture the request, preprocess log item to generate event log item and send the event log to target cloud server.

In Section 4.2.1, we mentioned that the preprocessing needs the contextual information of the log item, including the time and location of previously reached log item, to determine the continuity of the two continuous log items. To meet this requirement, we select one node from each LAN as *Local-Center Fog Node*, as shown in Fig. 3, which exists uniquely in each LAN of the fog

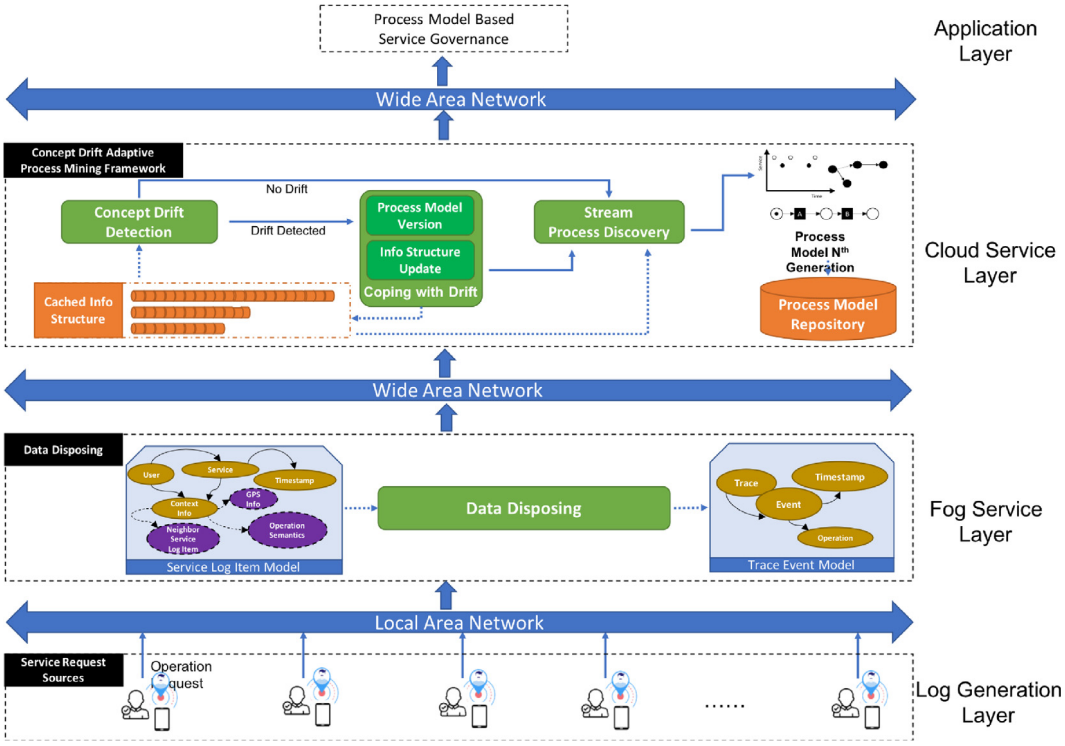


Fig. 1. Overview of fog computing based concept drift adaptive process mining framework.

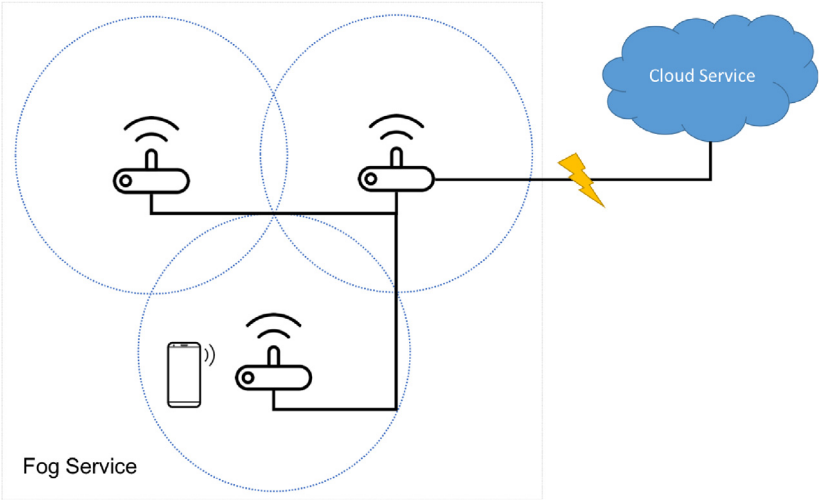


Fig. 2. Wireless internet service structure.

service layer and owns the function of log centralization, processing, saving, and submitting. All non-center fog nodes no longer execute preprocessing operations. Fog nodes in one wireless LAN can share messages rapidly through cables that link these nodes, as we have discussed before. Thus, as shown in Fig. 3, when fog nodes receive service log items that are uploaded by users, various types of fog nodes will act differently. Local-center fog node will preprocess and cache the log items, as well as refresh related continuity information of algorithm, while non-center fog nodes just transmit log items to the local-center fog node of the LAN.

In order to reduce time cost, local-center fog nodes will not transmit the standard event items to cloud servers immediately after preprocessing a newly arrived event item. After each predefined time interval, local-center fog nodes will package event logs, upload them to cloud servers, and then clean up its cache memory.

For the implementation of data preprocessing in streaming environment, we propose two strategies based on 4.2.1 section: optimistic strategy and pessimistic strategy, as shown in Table 1.

In Table 1, the latest observed service item (LOS) is the collection of newly cached service log items. When a service log item arrives, the local center fog node checks it firstly through the predefined entry service set and exit service set to determine whether it is an entry item, exit item or not-entry not-exit item. In case that an entry item arrives, if LOS is empty, then a new tracked case will be created, marking the newly arrived log item as start point and putting it into LOS. Optimistic strategy will assign the new case's ID to the corresponding event at once. Instead, pessimistic strategy will keep silent and wait. If LOS is not empty and the current log item pass the continuity check, then it will be marked as a waypoint of current tracked case and be put into LOS. Optimistic

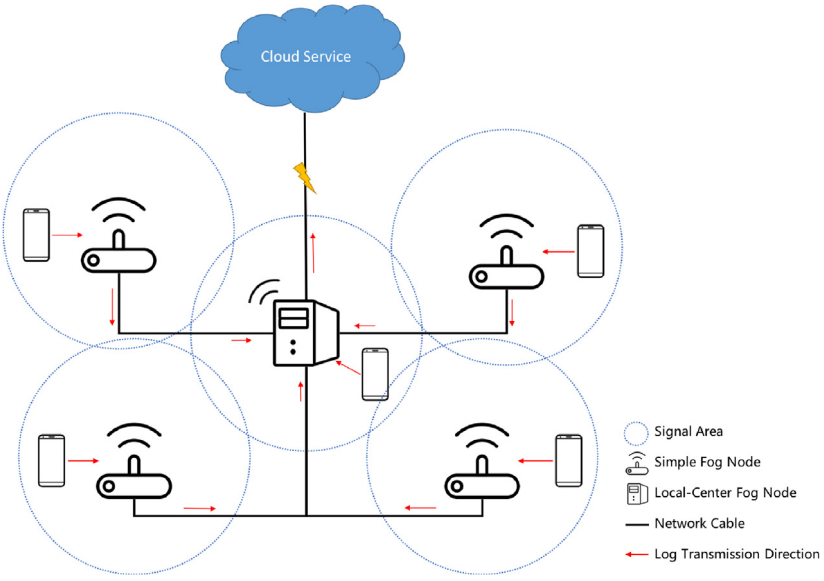


Fig. 3. The basic framework of fog computing.

Table 1
Rules on optimistic and pessimistic strategies.

Type check	LOS has service items	Continuity check	Condition	Action on LOS	Assign case ID	
					Optimistic	Pessimistic
Entry item	No	–	Continue Counting of Current Case (Start)	LOS ← service item	Return current case ID	Waiting
	Yes	Yes	Continue Counting of Current Case (Waypoint)	LOS ← service item	Return current case ID	Waiting
		No	Stop Counting of Current Case and Prepare for New Case, Continue Counting of Current Case (Start)	LOS ← 0 LOS ← service item	Return current case ID	Waiting
Exit item	No	–	–	Skip	Filtered	Filtered
	Yes	Yes	Continue Counting of Current Case (End), Stop Counting of Current Case and Prepare for New Case	LOS ← service item LOS ← 0	Return current case ID	Return current case ID to all items in LOS
		No	Stop Counting of Current Case and Prepare for New Case	LOS ← 0	Filtered	Filtered
Not-entry Not-exit Item	No	–	–	Skip	Filtered	Filtered
	Yes	Yes	Continue Counting of current Case (Waypoint)	LOS ← service item	Return current case ID	Waiting
		No	Stop Counting of Current Case and Prepare for New Case	LOS ← 0	Filtered	Filtered

strategy will assign current case's ID to the corresponding event, and pessimistic strategy still wait. If LOS is not empty and the continuity check is not passed, then the current case counting will be stopped. The LOS will be cleaned up and prepared to track the next new case. The log item will be marked as the start point and be put into LOS. The optimistic strategy will assign the new case's ID to the corresponding event. The pessimistic strategy will do nothing.

In case that an exit item arrives, if LOS is empty, then our method will skip the log item directly. If LOS is not empty and the current log item passes the continuity check, then it will be marked as an end point of current tracked case and be put into LOS. The optimistic strategy will assign the current case's ID to the corresponding event. The pessimistic strategy will assign the same ID to the corresponding events of all items in LOS, including the one just arrived. Then the LOS will be cleaned up to prepare for the next new case. If LOS is not empty and the continuity check is not passed, then the LOS will be cleaned up and there is no more operation.

In case that a not-entry not-exit item arrives, if LOS is empty at this time, then our method will skip this log item directly. If LOS is not empty and the current log item passes the continuity check, then it will be marked as a waypoint of current tracked case and put into the LOS. The optimistic strategy will assign the current case's ID to the corresponding event. Instead, the pessimistic strategy will wait. If the LOS is not empty and the continuity check is not passed, then the current case counting will be stopped and the LOS will be cleaned up to track the next new case. No item will be assigned with a case ID under these circumstances.

Fig. 4 is a comparison between optimistic and pessimistic strategy. It could be noticed that there are two differences between optimistic and pessimistic strategies. One is the maintenance of the LOS, the other is the method of case ID generation. In optimistic strategy, a newly arrived service log item just needs to make a judgment on continuity with its neighbor item, and we can decide whether to assign a case ID to it. Two cases are hence generated by optimistic strategy in Fig. 4. However, in pessimistic strategy, standard event will not be generated until an acceptable exit service arrives. In other words, pessimistic strategy only assigns case ID to log items in complete cases. For example, only C2 is generated in

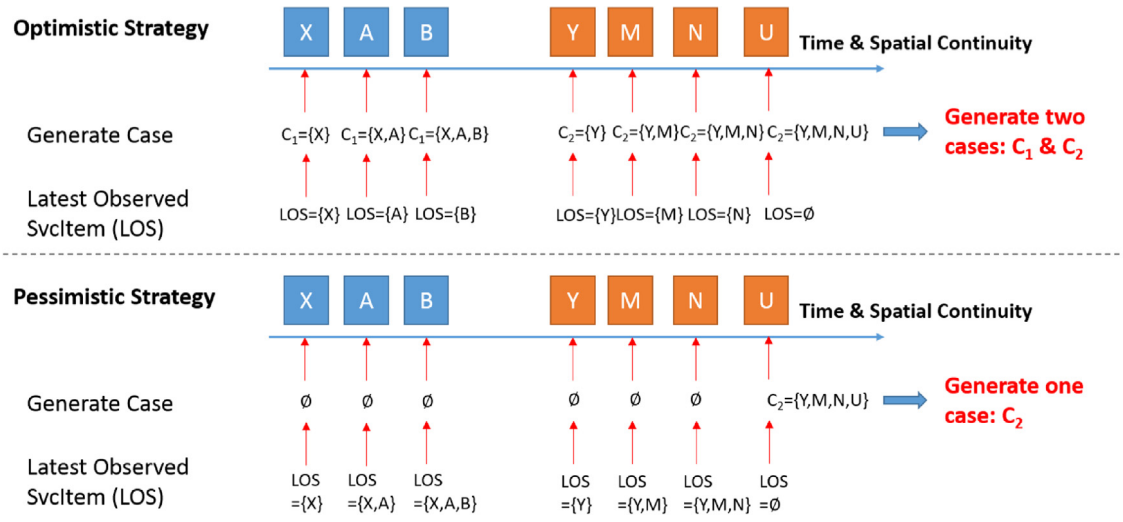


Fig. 4. Visual example of optimistic strategy and pessimistic strategy.

Fig. 4 because C1 is incomplete. Obviously, the Optimistic Strategy requires less memory and produces relatively more events with higher efficiency. But it may cause some false positives like C1 in Fig. 4 which is an incomplete trace but still obtains a case ID, because the event X, A and B pass the formal check of optimistic strategy. Traces generated by pessimistic strategy are correct in theory. But pessimistic strategy needs an additional memory. If service log items do not appear for a while, the LOS caught in the memory could not be moved out, regular scanning of the memory will be needed. In real-life application, we can choose optimistic or pessimistic strategy according to the actual requirements.

When logs are uploaded to cloud service layer from fog service layer, local fog center nodes gather and transmit standard log events that have been newly assigned with case IDs by both optimistic and pessimistic strategies.

4.3. Concept drift adaptive stream process discovery in cloud service layer

After data disposing in fog service layer, the service log items are converted to event logs and transmitted to the cloud service layer. However, log items are collected and preprocessed in a distributed way, fog service layer nodes in different LANs continuously and concurrently transmit these incomplete logs, such as some segments of different log traces, to cloud servers. As a result, the raw event logs in cloud service layer are chaotic. To address this problem, firstly we will aggregate these incremental log segments into a complete log stream according to each event's timestamp.

In order to mine log streams precisely, a concept drift adaptive approach is proposed to detect the concept drift actively, offering a more appropriate way for mobile applications.

Algorithm 1 defines the scheme of our concept drift adaptive process discovery method in general. After observing an event [Line 4–5], the features of the input event are extracted for further concept drift detection and stream process discovery [Line 6]. We store these features in *Cached Info Structure* [Line 7] and the concept drift is actively detected to judge the changing degree of the process model [Line 8]. Once the drift is figured out, our framework begins to cope with the changing process model. The current mined model is stored in *Process Model Repository* and tagged with a new version [Line 10], and the *Cached Info Structure* for discovering is updated as a new model [Line 11]. Whether the drift is detected or not, the current mined process model will be updated with the *Cached Info Structure* by the stream process discovery algorithm [Line 13].

Algorithm 1 Scheme for Concept Drift Adaptive Process Discovery from Service Log Item Stream

Input: S: Service Log Item Stream.

1: $P \leftarrow \emptyset$ /* Process Model Repository */
2: $p_{current} \leftarrow \emptyset$ /* Current mined process model */
3: **loop**
4: $svcItem \leftarrow observe(S)$ /* Service Log Item */
5: $e \leftarrow ctxConvert(svcItem)$ /* Context based conversion */
6: $features \leftarrow featureExtraction(e)$
7: Update Cached Info Structure with *features*
8: *detected* ← Concept Drift Detection with Cached Info Structure
9: **if** *detected* = *TRUE* **then**
10: $P \leftarrow p_{current}$
11: Update Cached Info Structure for Discovering New Model
12: **end if**
13: Update $p_{current}$ with Cached Info Structure
14: **end loop**

The implementation of Algorithm 1 in our framework is based on the stream declarative model discovery algorithm proposed in [14] and the online concept drift detection method declared in [21]. For stream declarative model discovery, Lossy Counting (LC) and Lossy Counting with Budget (LCB) algorithms could be applied to record the most frequent features from the point that starts to analyze till the current observed point to handle the storage limit and improve the mining efficiency [14], which is implemented using the *Cache Info Structure* in our framework. Then the traditional process mining algorithms, like α Algorithm, Declarative Miner Algorithm, etc. are adopted to mine the process model from different perspectives based on the *Cached Info Structure*. In the online concept drift detection method [21], statistical analysis is performed about the differences among α^+ relation features that are extracted in the detection window and the reference window. If the counting of continuous drifts is beyond a defined threshold, then a real drift point is taken into account according to statistical analysis. Therefore, the drifts can be figured out in incrementally recorded logs in [21], i.e. event streams, rather than based on completely recorded logs or traces.

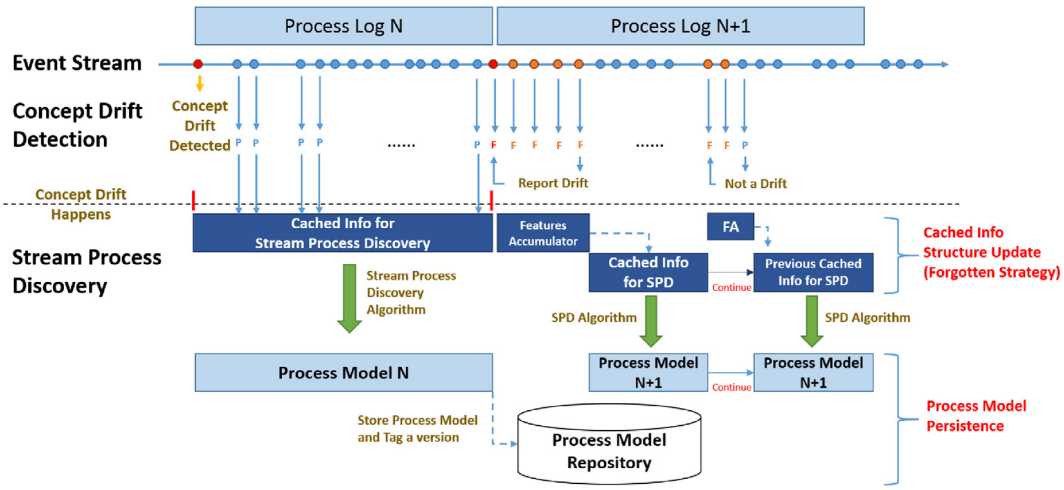


Fig. 5. Implementation for concept drift adaptive stream process discovery.

In our method, we focus on active concept drift detections while mining a process model and how to react when the drift is detected, thus offering a more suitable way for mobile applications with frequent releases. As shown in Algorithm 1 and Fig. 5, the concept drift is detected actively before the streams are mined. Our method defines two thresholds firstly, one is similarity threshold, and the other is statistical threshold. Drift point is marked when the observed event generates a model with lower similarity than the threshold. The method will check whether the drift points are temporary or persistent according to the statistical threshold. If the candidate of the drift point lasts beyond the statistical threshold, this candidate point will be reported as a real drift point, otherwise, it will be treated as an incident. For the events that pass statistical tests, the stream process discovery will begin immediately. For the events that fail in the tests, clean-up operations will be carried out to prepare for mining a new process model.

As shown in Fig. 5, when the algorithm has found a concept drift point candidate (in Fig. 5 marked with F), our method will stop the stream process discovery, store the extracted features in the *Feature Accumulator (FA)* temporarily, and wait for the statistical results of the coming events.

If un-continuity of event logs appears continuously after the candidate drift point, the candidate will be treated as a real concept drift point and considered as the start of a new process. The mined process model based on logs before concept drift point will be stored into the *Process Model Repository* and treated as historical version. A new *Cached Info Structure* instance will be created. The *Feature Accumulator* transfers the stored features of the events after the drift point to the *Cache Info Structure* and the stream process discovery will continue to find new process models.

The candidate point may be identified as an incident, i.e. some events after the candidate pass the statistical tests (in Fig. 5 marked with P). Under this circumstance, no drift of the process happens. The stopped stream discovery needs to be recovered. Before restarting, the accumulated features in *FA* will be transferred to the existing *Cached Info Structure*.

4.4. Multi-version process model based cloud service governance in application layer

With the running of the stream mining algorithm, multiple versions of the process model may be discovered from logs and stored in the Process Model Repository. By analyzing the evolution process of these models, we can obtain several cloud service governance guidelines, including service optimization, resource allocation, deployment optimization and performance monitoring.

(1) Service optimization. Because there exist gaps between APPs designers and users, modification of services always has unpredictable effects. In our framework the concept drift points are marked to indicate when changes occur in these processes. By analyzing the occurrence of these drifts, the changing period can be figured out. When analyzing the process changes caused by designers, for example, the mobile application developers, the changing period helps them to measure users' reaction time to the new release. For the drift caused by the APP users, by comparing the process models before and after the drift point, the changes of users' behaviors can be distinguished, which lead to better user profiles. Considering all changes of the processes as a whole, the evolution trace of the process models can be depicted. Therefore, based on the evolution traces, the next version of the model can be predicated for redesigning of the processes.

(2) Resource allocation. Most cloud computing service providers, such as Amazon, provide cloud services in multi-tenant modes where computing resources are rent by users concurrently depending on how much they actually need, so that the monitoring of multi-version process model evolution and the records of the concept drift points are referable for formulating cloud service governance strategies. For example, if we detect that the amount of some service requirements in current process model is less than the last version, we can consider reducing the amounts of hardware resources, which can minimize unnecessary expenses meanwhile not affect users' regular service requests.

(3) Deployment optimization. While more and more systems are deployed in a distributed architecture, services and databases are deployed in multiple physical hosts, and they access each other remotely through wide area networks. Systems can obtain not only faster calculation speed to respond to user's requests, but also stronger abilities to deal with failures, which are the benefits brought by distributed architectures. However, when a service deployed on one physical host accesses data stored on another host, it may cause a certain network time delay. Although this kind of delay is only 100 ms around, it is still totally perceptible, especially for transactions with high frequency. For time-sensitive mobile applications, time delay will affect user experience.

We try to improve time delay in some degree through multi-version process models. In cloud service application, there exist services with similar data sources. In m-health APP, for example, activities such as checking doctor list, checking doctor detail and making appointment with doctor need to make CRUD operations to the databases storing doctor information. We can deploy the services related to these activities and the databases storing doctor

information on similar or close physical hosts to reduce access time delay. By analyzing process model continuously, we can discover service groups with high frequency and similar data sources, and make deployment optimizations to reduce the average time delay.

(4) Performance monitoring. Services are usually requested in logical orders. For example, when you are going to make an appointment with a doctor using m-health APPs, you might have to make a decision about which hospital to go before selecting doctors. If one service crashes or performs poorly, it can affect more than one service, especially those who are central in the whole process. For this reason, once such a crash or failure occurs, maintenance must be conducted promptly. The concept drift adaptive stream process discovery mentioned above can realize real-time monitoring of systems. If a crash or failure appears in the system, the log stream mining function will change significantly, specifically the request frequencies of that broken service and its subsequent services will drop dramatically, which may lead to changes of the model structure. Maintenance personnel can catch these exceptions promptly with the help of the multi-version process model evolution monitoring, then check and repair in time so that the overall performance is effected as less as possible.

5. Experiment

In this section, several experiments are conducted to evaluate our methods proposed in this paper. The entire experimentation is conducted on a machine with 2.75 GHz Intel Core i5, equipped with 8G memory, Mac OS X 10.10.3, Java 8.

5.1. Datasets

Two datasets are prepared for the experiments. The logs are generated by CPN Tools 4.0. In CPN Tools, two variants of Make Appointment in an m-health APP are established (Fig. 6(a) and (b)).

In Fig. 6(a), when a user clicks the *Make Appointment* button on the home page, the appointing process starts. After several query requests, he/she will obtain the doctor's information and can select the appointment time. In this version, users can only make an appointment for free and pay for it when arriving at the hospital.

In Fig. 6(b), the APP designers add four features in 2.0 version with modifications on *Query* sub-process and *Payment* sub-process. Two search conditions are added when querying hospitals. When users get the department list, he/she can alternatively get the features of the departments directly, which is much more friendly to users. In addition, more information can be obtained in *Doctor Page* including patient assessments and rules of making an appointment. *Online Payment* and *Cancel Appointment* are also added in version2.0 as well. By CPN Tools, 1000 cases are generated for each process variant, with 15000 and 14525 events respectively.

5.2. Evaluations on log disposing

In this section, we evaluate our data disposing method described in Section 4.2. Before evaluation, the *Case IDs* of the log items are erased, which is the prerequisite for our method. Some noises, which accounts for 3% of all logs approximately, are added to the datasets, such as removing the entry item or exit item of one case. These noises can help make quantitative evaluation for the correctness of our method. Datasets used in the experiment contain 227965 service log items.

We choose F_1 *measure* calculated based on the *precision* and *recall* metrics for the quantitative measure of the experiment. More specifically, results disposed in our log will be categorized into four groups:

Table 2
F_1 *measure* on optimistic and pessimistic strategies.

Metric	Optimistic Strategy	Pessimistic Strategy
precision	0.886	1.0
recall	0.754	0.754
F_1	0.815	0.860

- True Positive (TP): correctly tagged service log items, comparing with the generated Case IDs in the synthetic log.
- False Positive (FP) refers to incorrectly tagged service log items, i.e. incomplete cases.
- True Negative (TN) refers to correctly missed service log items, such as noises in the logs.
- False Negative (FN) refers to incorrectly missed service log items. In most cases, this is because inappropriate δ_{time} and δ_{space} are chosen.

The experimental results are shown in Table 2. The F_1 of Optimistic Strategy is 0.815, lower than the F_1 of Pessimistic Strategy, which is 0.860. The Optimistic Strategy makes immediate decision when a new service log item arrives under the assumption that the exit service log item will arrive in the coming stream, as mentioned in Section 4.2.2. This will cause more False Positives (FP), resulting in a lower precision value. The *precision* of Pessimistic Strategy is 1.0 because it barely assigns the case ID to all service log items of the complete case. The *recall* values of these two strategies are 0.754. It is identical that both of these two strategies correctly tag the cases that pass the Continuity Checks in Section 4.2.1 but incorrectly miss the cases that fail the Continuity Checks. In other words, the values of TP and FN are the same. False Negatives have strong relationships with δ_{time} and δ_{space}. It is recommended that these two thresholds are selected based on the average operation time and space interval of users from historic data.

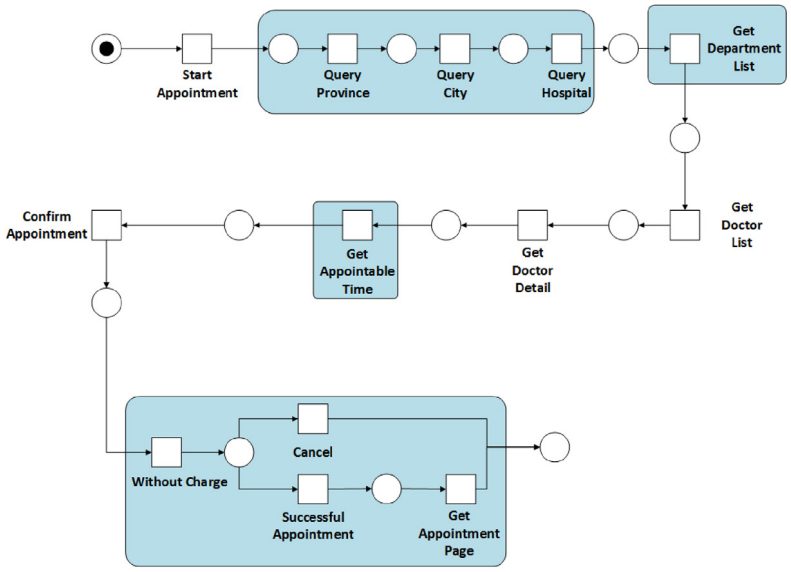
Fig. 7(a) and (b) show the numbers of output events in Optimistic Strategy and Pessimistic Strategy. We assume that the service log item is arriving at a constant speed. Fig. 7(a) is the fragments of the first 1000 service log items. The dash line indicates the Pessimistic Strategy, while the solid line refers to the Optimistic Strategy. It is clear that the Optimistic Strategy can maintain a relatively stable output rate. When a service log item arrives, the case identifier will be assigned in most cases and the number of output events are approximately 1 for each service log item. But for the Pessimistic Strategy, the number of output events varies for each coming service log item. As is shown in Table 1, the Pessimistic Strategy will generate identifiers only when the exit service log item is observed. So peaks will occur when exit service log items arrive and pass the continuity checks. The peak values are the number of events of a complete case that are stored in LOS.

5.3. Evaluations on concept drift adaptive process discovery

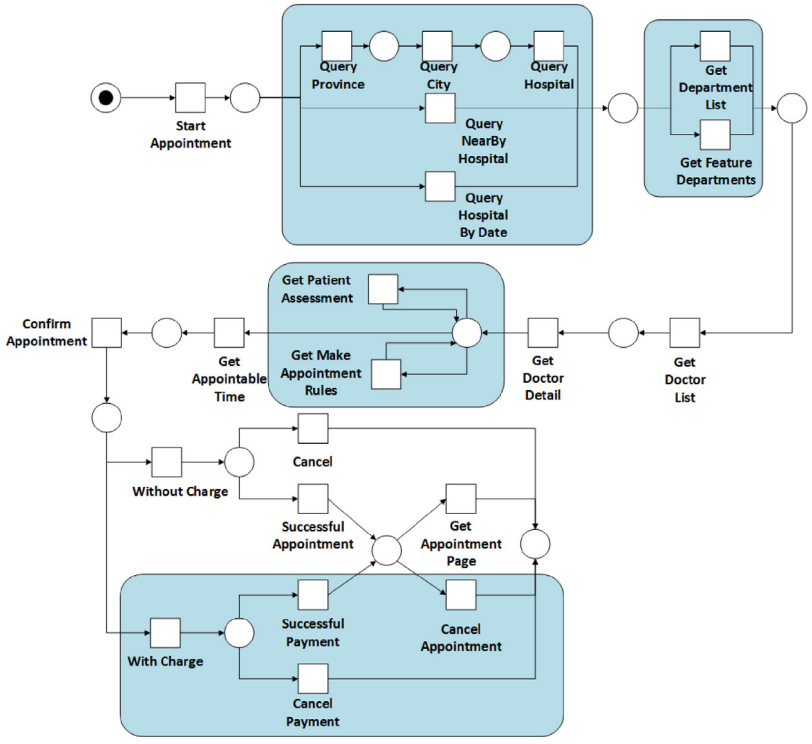
In this Section, we conduct experiments on the concept drift adaptive process discovery approach discussed in Section 4.3. To simulate the concept drifts in the *Make Appointment* process, we merge the synthetic logs $\mathcal{L}_1$ and $\mathcal{L}_2$ generated by process variants in Fig. 6(a) and (b) with eight alternations of $\mathcal{L}_1$ and $\mathcal{L}_2$ to create fifteen concept drift points. The entire dataset contains 236200 events.

Two criteria are chosen to evaluate our approach. One is the F_1 *measure* used to evaluate the effectiveness of our method, the other is the *performance measure* applied to evaluate the efficiency of our method. Both of these two evaluations will be performed in comparison with our approach with the method presented in [14].

Like stream declarative model discovery algorithm, the golden standard in our experiment is the constraints (i.e. $\mathcal{M}_1$ and $\mathcal{M}_2$)



(a) v1.0



(b) v2.0

Fig. 6. Make Appointment in an m-health APP (rounded rectangles indicate the change of models).

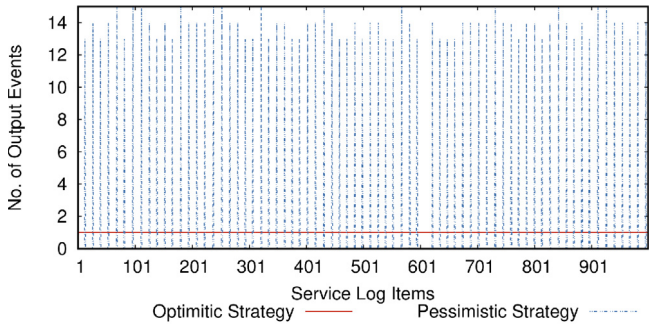
mined from the static log $\mathcal{L}_1$ and $\mathcal{L}_2$ by Declare Maps Miner Plugin [41] in ProM 6.5.12. The experimental results are shown in Fig. 8.

As is shown in Fig. 8, our method produces a relatively stable $F1$ value compared with the original stream declarative model discovery algorithm and obtains a better mining result when concept drift happens. Light gray areas in Fig. 8(a) cover 5000 events after each drift point. It is clear that the $F1$ value in our method increases faster than the original stream declarative model discovery algorithm and reaches a relatively high value (0.8).

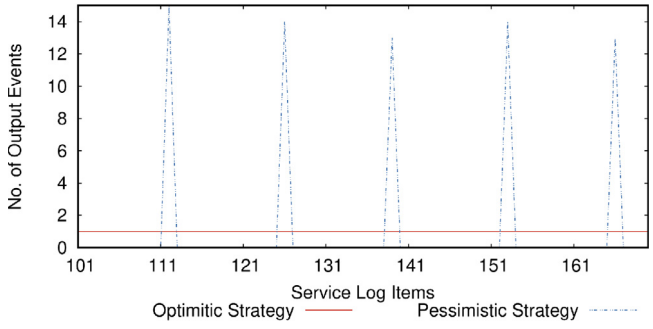
Table 3 demonstrates the $F1$ improvements after each drift point. We cover 5000 events and set the evaluation interval to 50 events and get 100 $F1$ values for each drift point. Among 15 drift points, our approach gets higher $F1$ values at 75.467% evaluation points. And the average $F1$ improvement percentage is 89.053%.

Table 3
*F*1 improvement after concept drift point detection.

Drift point no.	F1 improvement	
	F1 improvement points percentage (%)	Average F1 improvement percentage (%)
1	53	4.843
2	89	193.220
3	79	16.452
4	90	146.435
5	55	8.300
6	96	168.496
7	60	21.261
8	93	145.544
9	64	24.788
10	93	225.487
11	62	24.712
12	91	151.054
13	60	20.128
14	92	172.161
15	55	12.912
Average	75.467	89.053

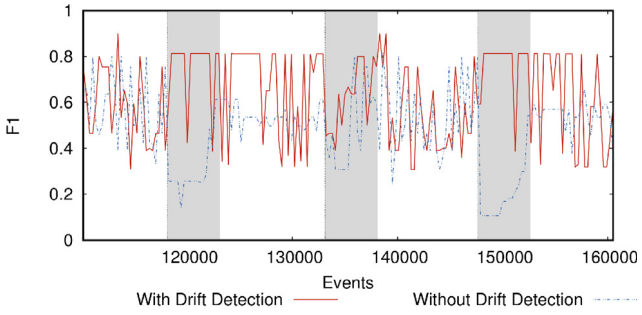


(a) The First 1000 Service Log Items

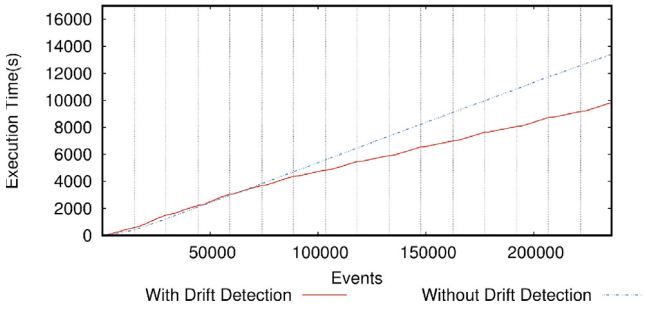


(b) 101th to 170th Service Log Items

Fig. 7. Number of events produced when each service log item arrives under Optimistic and Pessimistic Strategies.



(a) F_1



(b) execution time

Fig. 8. Two metrics on concept drift adaptive process discovery. (Gray dotted vertical line in each subfigure indicates the concept drift points detected in our approach. And light gray area in Fig. 8(a) covers 5000 events after each drift point.)

This indicates that concept drift points help the improvement of the effectiveness for stream process discovery.

The execution time of these two methods are shown in Fig. 8(b). Our approach keeps a linear relationship with data size. This advantage is more obvious when the size of event data increases. This indicates that the added concept drift detections cost little time and have a small impact on the efficiency of the stream process discovery. And the cleaning of events, which belongs to the previous model at drift points, offers a better dataset that contains more events belonging to the new model for further stream process discovery, which reduces the complexity of the mining task.

5.4. Case study

Our approach has been applied in the cloud service governance of a Chinese m-health APP, which provides practical advice on system monitoring and APP UI redesigns based on the real-time user behavior models.

As is mentioned above, our approach can figure out changes of the process model (i.e. the user behavior model in mobile APPs), and discover the up-to-date model by streaming analysis. After the drifts are detected, the previous model will be stored in the *Process Model Repository* for further analysis. The overview of System Monitoring Platform based on our approach is demonstrated in Fig. 9. By comparing with the previous version of the process model in *Process Model Repository*, the new feature, such as *Get*

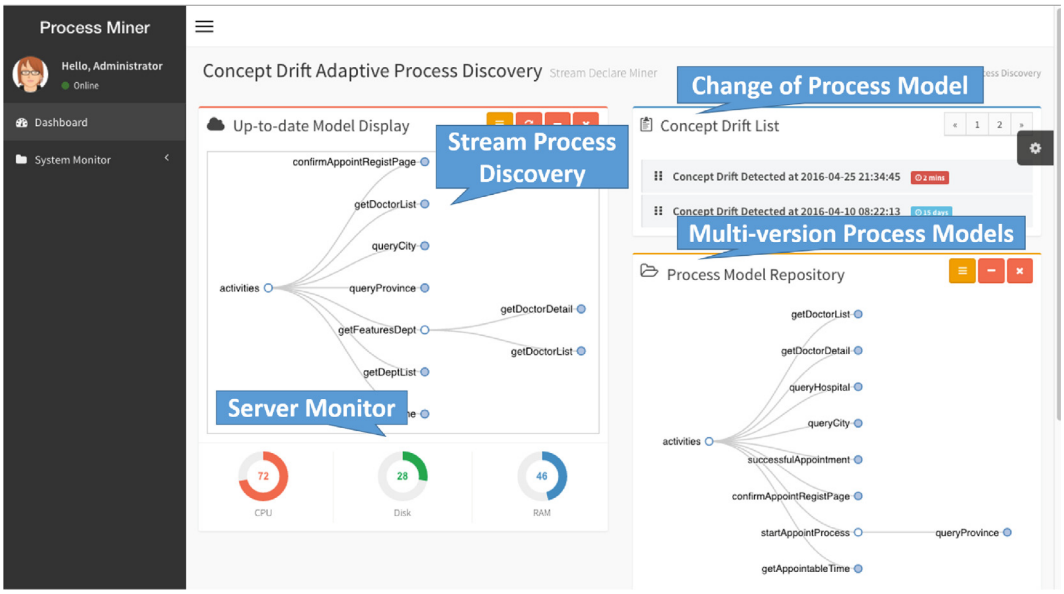


Fig. 9. Overview of System Monitoring Platform based on concept drift adaptive process mining.

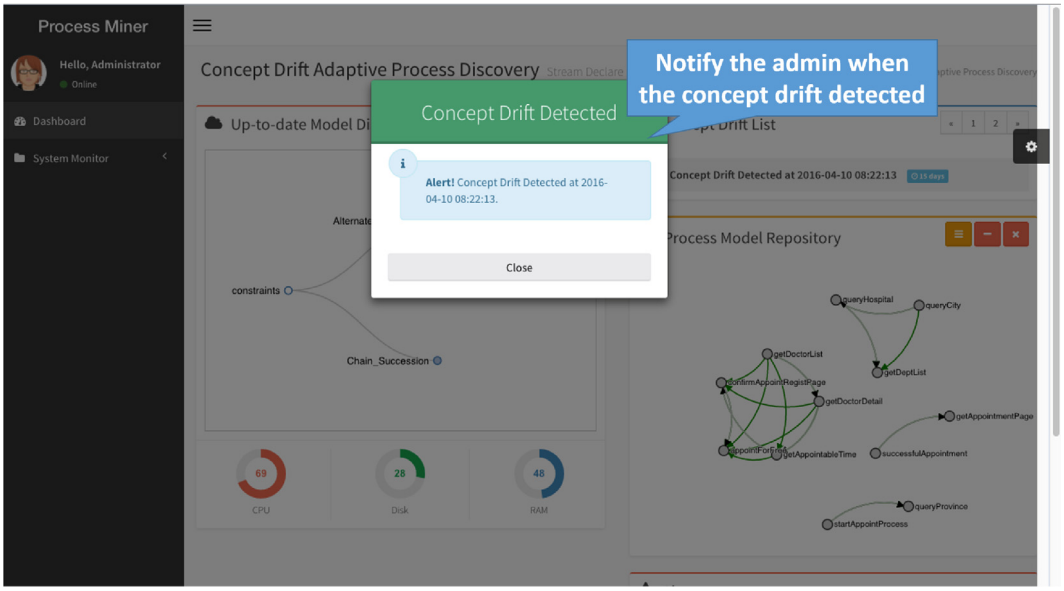


Fig. 10. Notify administrators of concept drift detected.

Feature Departments, is included in the up-to-date model. Besides, the concept drift points are gathered to show the frequency of changes of the user behavior model. In this APP, concept drifts occur mainly after the new version release. After the concept drift is detected, a notification will be sent to the administrator for detailed judgment why the concept drift happens.

As is shown in Fig. 10, these concept drift points and process models help the APP developers to analyze how the new APP releasing influences users. Fig. 9 indicates that the redesign of APP has a positive effect on users' behaviors, by providing more search options for users and improving their experience. However, some modifications incorrectly change users' behaviors and these can be distinguished in our approach to assist with the redesign of APP UI in new releases. Moreover, from the view point of service administrators, cloud resource optimization can be considered with each process model alternation where the proportion of calls for services has changed, which can improve the stability of the whole server while reduce costs as much as possible.

In addition, the platform keeps records of the standard behavior model extracted from the majority of users. These can be used for real-time analysis on certain users. In current version, the majority of users spent less than 5 min on the *Make Appointment* process in this APP (covers 89% traces). When a user is executing this process, if he/she stays on the *Confirm Appointment* page for 2 min, he/she exceeds the average time (30 s). By the real-time standard user behavior model, the following activities of the *Confirm Appointment* can be identified and messages will be sent to the user for help. In this APP, some tips will pop up to guide the user for following operations. If the user is detected to complete the *Make Appointment* process very fast and frequently in a short period, he/she will be marked as a potential risk user, and a captcha will be added in the following essential functions, such as *Confirm Appointment*, to slow down the operating speed for risk managements.

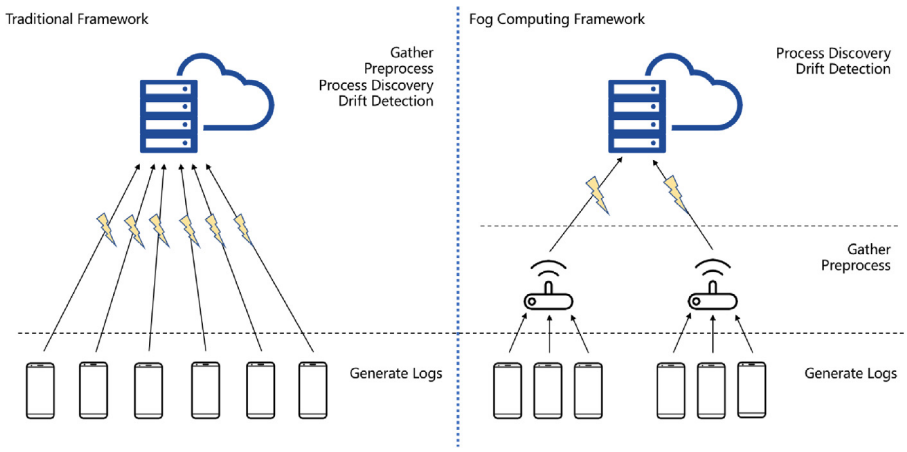


Fig. 11. Visual comparison between traditional framework and fog computing framework.

6. Discussion

In this paper, we propose a novel fog computing based concept drift adaptive stream process discovery framework. In this section, we will make discussions in fog computing framework and our process discovery method.

6.1. Fog computing framework

In our method, fog computing is mainly applied for logs gathering and preprocessing in the fog service layer of Fig. 1. For discussion, we compare our method with traditional framework where cloud server is responsible for log preprocessing.

As is shown in Fig. 11, in traditional framework, applications will transmit corresponding log items to cloud server immediately through wide area network as soon as users operate. Cloud server gathers and preprocesses all raw log items, and then marks them as the input of our concept drift adaptive process discovery algorithm. Different from this traditional framework, our fog computing based approach moves some system operations closer to user's local area network. Fog nodes (routers) in LAN gather and preprocess raw log items, and store them temporarily. Fog nodes package and transmit these standard event logs timely to cloud server, which can execute concept drift adaptive process discovery algorithm directly with no need for other operations.

With above comparison, we can conclude several advantages of our framework:

(1) Lower overhead. A piece of log will be created every time when a user does an operation in the application. Therefore, in traditional framework, log transmission happens extremely frequently. This kind of practice will produce lots of time delay and performance overhead, especially for those applications with huge quantity of users. The cloud server handles stream data in a single thread, so frequent log uploading will not only cause transmission delay and take up network bandwidth, but may also incur data congestion on the server side, making the efficiency of process discovery and drift detection algorithm lower. According to Section 4.3, process discovery and drift detection algorithm will not be executed on cloud servers for every newly arrived log item, so the overhead previously discussed is avoidable. In our fog computing framework, raw log items only flow in LAN which is closer to users after generated by applications, whose transmission delays are almost negligible. After preprocessing in fog nodes, standard event logs will not be uploaded immediately but stored temporarily and packaged to be transmitted after a period of time. Assuming that there are N raw log items generated in a time segment, by applying our fog computing framework, the total transmission time and server concurrency can be reduced to about 1/N times of traditional framework. Obviously, our implementation method greatly cuts back the load pressure on cloud servers, and decreases total transmission time and performance overhead, so that the system as a whole can run more stably and efficiently.

(2) More precise preprocessing. The preprocessing algorithm introduced in Section 4.2 performs a continuity check of a raw log item with its neighbors to determine if this log item belongs to the current tracked case. In the traditional framework, if multiple mobile application instances generate and upload log items to the cloud server at the same time, the accuracy of the output of this preprocessing algorithm in the cloud is apparently very awful, because this algorithm cannot handle the situation where events belonging to multiple different cases alternately come. However, in our fog computing framework, there is no interference between instances running in different LANs because log items will not leave the LAN until they are preprocessed. In other words, after applying the fog computing framework, the algorithm in Section 4.2 obtains a certain of ability to process data concurrently, which greatly improves the accuracy of preprocessing results.

It is totally feasible to implement our fog computing framework in real life. In fact, expanding on the LAN structure in Fig. 2 and using virtualization technology (such as container) to encapsulate and deploy fog service will result in a highly scalable general-purpose fog computing network that satisfies our method. Container has the features of lightweight, easy-deployment, isolation and portability [42], therefore, multiple containers can run simultaneously on resource-constrained hardware, each of which holds one fog service.

Internet Service Provider (ISP) deploys wireless network access points at various locations and connects them within certain areas to construct some LANs, as is shown in Fig. 12. Some service providers, like Alibaba.com, can rent LANs from ISPs to construct the Internet communication and set up the Local Center Fog Nodes in each LAN, and then rent memory or other computing resources to users who deploy fog computing applications. The Simple Fog Nodes are better be owned by the fog computing users because the simple fog nodes are closely related to devices which are owned by end users. Under this structure, various fog services can be encapsulated by service providers into exclusive containers and deployed in fog nodes independently on demand.

6.2. Concept drift adaptive stream process discovery

Usually, process discovery methods are divided into two categories: online discovery and offline discovery. In this part, to

Table 4
Comparison among process discovery methods.

Features	Our approach	Online process discovery [14]	Offline process discovery [41]
Main objective	To achieve comprehensive and continuous user behavior analysis and service management in mobile APPs	To discover process model from event streams	To discover process model from event logs
Main method	Active concept drift detection + Stream process discovery	Stream process discovery	Static analysis on event logs
Period	Online/Execution time	Online/Execution time	Offline/Design time
Input	Service log stream	Event stream	Static event log
Log quality	Low (Missing Case ID & multiple process instances in one log)	High (standard event stream)	High (standard event log)
Data disposing	Context data based log conversion	Not considered	Not considered
Output	Multiple versions of process model	Single process model	Single process model
Concept drifts	Detected while discovering the process model	Figured out by process model conformance checking	Not considered
Performance	Fast	Fast	Slow

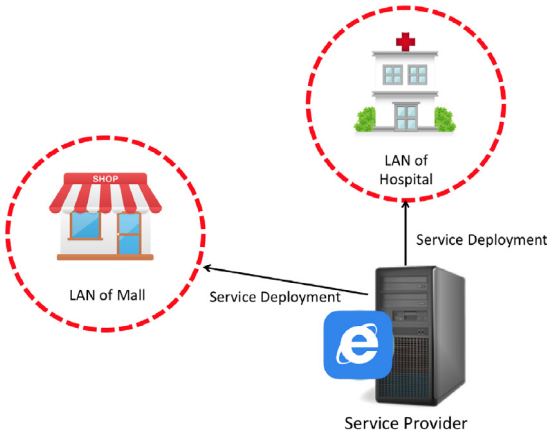


Fig. 12. Fog service deployment.

demonstrate the characteristics of our approach, an online discovery method [14] which discovers the declarative model from event streams and an offline discovery method [41] which finds the declarative model from static event logs are selected to be compared with our approach. The comparison results are shown in Table 4.

The method presented in this paper combines active concept drift detection and stream process discovery. Detections will be performed before stream process discovery while the other online method only takes stream process discovery into account and the offline method carries out a static analysis on event logs. Our approach and [14] are two forms of stream process discovery, which both can be applied at execution time. But [41] is an offline analysis which requires the event logs to have been accumulated and will be performed at design time in most cases. Obviously, the inputs of our approach and other online process discovery methods are streaming data while the inputs of offline discovery methods are event logs. Because of the characteristics in mobile APPs, the log streams collected are often service log streams which are incomplete. So a context based log conversion method is reported in this paper to adapt service logs to standard event logs by case ID completions. Unlike other online and offline process discovery algorithms, our approach will generate multiple versions of the process model for further service governance. Concept drifts will be detected actively rather than figured out by process model conformance checking in other online algorithms. Although concept drift detections will be performed when every event arrives, the performance is better than other offline algorithms. Sometimes, our clean-up operations after drift points help the stream process discovery to improve its efficiency and obtain a better performance.

To sum up, our approach successfully handles the incompleteness of logs in mobile applications by deducing the essential *Case ID* fields and distinguishing different versions of the process model. Better mining results are obtained after concept drifts occur, providing more accurate user behavior models to support comprehensive analytical goals in most mobile APPs.

7. Conclusion

A fog computing based concept drift adaptive process mining framework is proposed in this paper to solve the log incompleteness problem and to provide process model evolution analysis in mobile applications. The approach firstly gathers the service logs of mobile applications in the fog service layer, where fog nodes convert them to standard event logs based on context data by field mappings and case identifier completion. We also demonstrate the implementation of a fog computing based log conversion method in streaming environment and offer both optimistic and pessimistic strategies about the case identifier completion. After the data disposing on low-quality service logs, standard event logs are transmitted to the cloud service layer and concept drift adaptive stream process discovery is carried out. Due to the changing process model caused by frequent releases in mobile applications, an active concept drift detection is performed before stream process discovery and the clean-up operations on data structures will be carried out after the drift point is figured out. This leads our framework to sense and mine the up-to-date model more promptly, providing a more appropriate way to monitor the evolution of real-life process models for mobile applications. Our framework collects the mining results and tags them with versions, which can be considered for the implementation of more effective cloud service governance.

As for the future work, we will extend the implementation of our framework with more perspectives, such as social networks, control-flow models, etc. to provide more comprehensive mining results. Then it will be considered to add business-rule-related information in the temporal and spatial model to deal with the situation where two events from different processes but happening almost at the same time are identified as the same case ID in the same fog node. Meanwhile we will design low-complexity noise processing algorithms or frequency-based mining method to reduce the number of missed logs in Fog Service Layer. In addition, more experiments will be conducted on real-life logs to verify the practicability of our framework. And as some big data analytical techniques also take into consideration the needs for real-time computing [5,43], big data frameworks such as Spark, Storm, etc. will be introduced into our approach.

Acknowledgments

This research is supported by the National Natural Science Foundation of China under Grant No. 61373030, and the Science and Technology Key Project of Zhejiang Province of China under Grant 2017C02036.

References

[1] W.M.P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, first ed., Springer Publishing Company, Incorporated, 2011.
[2] A.J.M.M. Weijters, W.M.P. van der Aalst, Rediscovering workflow models from event-based data using little thumb, Integr. Comput.-Aided Eng. 10 (2) (2003) 151–162.
[3] T. Molka, D. Redlich, M. Drobek, A. Caetano, X.J. Zeng, W. Gilani, Conformance checking for BPMN-based process models, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC'14, ACM, New York, NY, USA, 2014, pp. 1406–1413.
[4] W. van der Aalst, Service mining: Using process mining to discover, check, and improve service behavior, IEEE Trans. Serv. Comput. 6 (4) (2013) 525–535.
[5] Hongming Cai, Boyi Xu, Lihong Jiang, Athanasios V. Vasilakos, IoT-based big data storage systems in cloud computing: Perspectives and challenges, IEEE Internet Things J. 4 (1) (2017) 75–87.
[6] A. Enayet, M.A. Razzaque, M.M. Hassan, A. Alamri, G. Fortino, A mobility-aware optimal resource allocation architecture for big data task execution on mobile cloud in smart cities, IEEE Commun. Mag. 56 (2) (2018) 110–117.
[7] R. Gravina, C. Ma, P. Pace, G. Aloi, W. Russo, W. Li, G. Fortino, Cloud-based Activity-aaService cyberphysical framework for human activity monitoring in mobility, Future Gener. Comput. Syst. 75 (2017) 158–171.
[8] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, C. Savaglio, Enabling iot interoperability through opportunistic smartphone-based mobile gateways, J. Netw. Comput. Appl. 81 (2017) 74–84.
[9] W. van der Aalst, T. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1128–1142.
[10] A.K.A. de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, A.J.M.M. Weijters, Process Mining: Extending the α-Algorithm to Mine Short Loops, Eindhoven University of Technology, Eindhoven, 2004.
[11] L. Wen, W.M.P. van der Aalst, J. Wang, J. Sun, Mining process models with non-free-choice constructs, Data Min. Knowl. Discov. 15 (2) (2007) 145–180.
[12] A. Weijters, W.M.P. Aalst, A.A.K. Medeiros, Process Mining with the Heuristics Miner-Algorithm, Vol. 166, 2006.
[13] A. Burattin, A. Sperduti, W.M.P. van der Aalst, Heuristics miners for streaming event data, CoRR abs/1212.6383.
[14] A. Burattin, M. Cimitile, F.M. Maggi, A. Sperduti, Online discovery of declarative process models from event streams, IEEE Trans. Serv. Comput. 8 (6) (2015) 833–846.
[15] M. Hassani, S. Siccha, F. Richter, T. Seidl, Efficient process discovery from event streams using sequential pattern mining, in: 2015 IEEE Symposium Series on Computational Intelligence, 2015, pp. 1366–1373.
[16] D. Redlich, T. Molka, W. Gilani, G.S. Blair, A. Rashid, Scalable dynamic business process discovery with the constructs competition miner, in: SIMPDA, Citeseer, 2014, pp. 91–107.
[17] R.P.J.C. Bose, W.M.P.V.D. Aalst, I. liobait, M. Pechenizkiy, Handling concept drift in process mining, Lecture Notes in Comput. Sci. 4 (1) (2011) 391–405.
[18] R.P.J.C. Bose, W.M.P. van der Aalst, I. liobait, M. Pechenizkiy, Dealing with concept drifts in process mining, IEEE Trans. Neural Netw. Learn. Syst. 25 (1) (2014) 154–171.
[19] J. Carmona, R. Gavaldà, Online techniques for dealing with concept drift in process mining, in: Proceedings of the 11th International Conference on Advances in Intelligent Data Analysis, IDA'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 90–102.
[20] F.M. Maggi, A. Burattin, M. Cimitile, A. Sperduti, Online process discovery to detect concept drifts in ltl-based declarative process models, in: On the Move To Meaningful Internet Systems: OTM 2013 Conferences: Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Graz, Austria, September 9-13, 2013. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 94–111.
[21] A. Ostovar, A. Maaradji, M. La Rosa, A.H.M. ter Hofstede, B.F.V. van Dongen, Detecting drift from event streams of unpredictable business processes, in: Conceptual Modeling: 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings, Springer International Publishing, Cham, 2016, pp. 330–346.
[22] D.R. Ferreira, D. Gillblad, Discovering process models from unlabelled event logs, in: Business Process Management: 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 143–158.
[23] M. Walicki, D.R. Ferreira, Sequence partitioning for process mining with unlabeled event logs, Data Knowl. Eng. 70 (10) (2011) 821–841.
[24] D. Bayomie, I.M.A. Helal, A. Awad, E. Ezat, A. ElBastawissi, Deducing case IDs for unlabeled event logs, in: M. Reichert, H.A. Reijers (Eds.), Business Process Management Workshops, Springer International Publishing, Cham, 2016, pp. 242–254.
[25] I.M.A. Helal, A. Awad, A.E. Bastawissi, Runtime deduction of case ID for unlabeled business process execution events, in: 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications, AICCSA, 2015, pp. 1–8.
[26] Q. Zeng, F. Lu, C. Liu, H. Duan, C. Zhou, Modeling and verification for cross-department collaborative business processes using extended petri nets, IEEE Trans. Syst. Man Cybern. Syst. 45 (2) (2015) 349–362.
[27] B. Li, S. Ji, D. Qiu, H. Leung, G. Zhang, Verifying the concurrent properties in bpel based web service composition process, IEEE Trans. Netw. Serv. Manag. 10 (4) (2013) 410–424.
[28] M. Montali, M. Pesic, W.M.P.v.d. Aalst, F. Chesani, P. Mello, S. Storari, Declarative specification and verification of service choreographiess, ACM Trans. Web 4 (1) (2010) 3:1–3:62.
[29] L.V. Allen, D.M. Tilbury, Anomaly detection using model generation for event-based systems without a preexisting formal model, IEEE Trans. Syst. Man Cybern. - Part A: Syst. Hum. 42 (3) (2012) 654–668.
[30] B. Xu, L. Xu, H. Cai, L. Jiang, Y. Luo, Y. Gu, The design of an m-health monitoring system based on a cloud computing platform, Enterp. Inf. Syst. 11 (1) (2017) 17–36.
[31] W. Li, I. Santos, F.C. Delicato, P.F. Pires, L. Pirmez, W. Wei, H. Song, A. Zomaya, S. Khan, System modelling and performance evaluation of a three-tier Cloud of Things, Future Gener. Comput. Syst. 70 (2017) 104–125.
[32] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Edition of the Mcc Workshop on Mobile Cloud Computing, 2012, pp. 13–16.
[33] L.M. Vaquero, L. Rodero-Merino, Finding your way in the fog: Towards a comprehensive definition of fog computing, SIGCOMM Comput. Commun. Rev. 44 (5) (2014) 27–32.
[34] I. Stojmenovic, S. Wen, The fog computing paradigm: Scenarios and security issues, in: Computer Science and Information Systems, 2014, pp. 1–8.
[35] T.H. Luan, L. Gao, Z. Li, Y. Xiang, L. Sun, Fog computing: Focusing on mobile users at the edge, CoRR abs/1502.01815.
[36] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, K. Mankodiya, Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare, Future Gener. Comput. Syst. 78 (2018) 659–676.
[37] J. Zhu, D.S. Chan, M.S. Prabhu, P. Natarajan, H. Hu, F. Bonomi, Improving web sites performance using edge servers in fog computing architecture, in: 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, 2013, pp. 320–323.
[38] M. Aazam, E.N. Huh, (2014) Fog computing and smart gateway based communication for cloud of things, in: International Conference on Future Internet of Things and Cloud, 2014, pp. 464–470.
[39] Y. Wang, T. Uehara, R. Sasaki, Fog computing: Issues and challenges in security and forensics, in: Computer Software and Applications Conference, 2015, pp. 53–59.
[40] R. Roman, J. Lopez, M. Mambo, Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges, Future Gener. Comput. Syst. 78 (2018) 680–698.
[41] F.M. Maggi, Declarative process mining with the declare component of ProM, in: Bpm Demos, 2013.
[42] A. Zanni, A. Zanni, Feasibility of fog computing deployment based on docker containerization over raspberrypi, in: International Conference on Distributed Computing and NETWORKING, 2017, p. 16.
[43] S. Marchal, J. Franois, R. State, T. Engel, Phishstorm: Detecting phishing with streaming analytics, IEEE Trans. Netw. Serv. Manag. 11 (4) (2014) 458–471.

Tao Huang received his B.S. degree in Software Engineering in University of Electronic Science and Technology of China, Chengdu, China, in 2017. He is now a postgraduate student in School of Software, Shanghai Jiao Tong University.

Boyi Xu received the B.S. degree in industrial automation and the Ph.D. degree in management science from Tianjin University, Tianjin, China, in 1987 and 1996, respectively. Currently, he is an Associate Professor at the College of Economics and Management, Shanghai Jiao Tong University, Shanghai, China. His research interests include enterprise information systems, Internet of things, and business intelligence.

Hongming Cai received his B.S. degree, M.S. degree and Ph.D. degree from Northwestern Polytechnical University, China in 1996, 1999 and 2002, respectively. Now he is a professor in School of Software, Shanghai Jiao Tong University, China. He is a standing director of China Graphics Society, a senior member of ACM/IEEE, and a senior member of China Computer Federation. He is rewarded as "National outstanding scientific and technological worker" by China Association for Science and Technology in 2012.

Jiawei Du was born in Sichuan Province, China in 1993. She received the B.S. degree in software engineering from Shanghai Jiao Tong University, China, in 2015, now she is a postgraduate student in School of Software, Shanghai Jiao Tong University, China.

Kuo-Ming Chao obtained his MSc and PhD degrees from Sunderland University, UK in 1993 and 1997 respectively. From 1997, he worked at Engineering Design Centre in Newcastle-upon-Tyne University as research associate for more than 3 years before he joined Coventry University as senior lecturer in 2000. Between 2007 and 2008, he joined British Telecom Research Lab as short term research fellow. His research interests include the areas of intelligent agents, service-oriented computing, cloud computing and big data etc. as well as their applications such as energy efficiency management and green manufacturing etc.

Chengxi Huang received his BS degree from Shanghai Jiaotong University, China, in 2010. He is currently pursuing the Ph.D. degree at Information System Technology Lab, School of Software, Shanghai Jiaotong University.

His research interests include model driven engineering, service computing.