# An Approach to Modeling and Discovering Event Correlation for Service Collaboration

Meiling Zhu[1,2,3(✉)], Chen Liu[2,3], Jianwu Wang[4], Shen Su[2,3], and Yanbo Han[2,3]

[1] School of Computer Science and Technology, Tianjin University, Tianjin 300350, China
meilingzhu2006@126.com

[2] Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data, North China University of Technology, Beijing 100144, China
{liuchen, sushen, hanyanbo}@ncut.edu.cn

[3] Cloud Computing Research Center, North China University of Technology, Beijing 100144, China

[4] Department of Information Systems, University of Maryland, Baltimore County, Baltimore, MD 21250, USA
jianwu@umbc.edu

**Abstract.** In an IoT (Internet of Things) environment, event correlation becomes more complex as events usually span over many interrelated sensors. This paper refines event correlations in an IoT environment. We extend our previous service hyperlink model to encapsulate such event correlations. To effectively discover service hyperlinks, we transform the event correlation discovery problem into a frequent sequence mining problem and propose *CorFinder* algorithm. Moreover, we apply our approach to improve anomaly warning in a power plant instead of simulation. Besides the application, we have made extensive experiments to verify the effectiveness of our approach.

**Keywords:** IoT service · Service hyperlink · Sensor event · Event stream · Event correlation

## 1 Introduction

Nowadays, sensors are widely deployed in industrial environments to monitor devices' status in real-time. A sensor continuously generates sensor events and a series of sensor events are correlated with each other. Such event correlations are modeled to enable application-level sensor collaboration. We designed an event log based on a stream data processing infrastructure [1, 2]. However, the correlations can be dynamically interwoven, and data-driven analysis would be of help.

Event correlation discovery problem is concerned about how to identify relationships among sensor events. Similar research has also received notable attention for the discovery, monitoring and analysis of processes. In those studies, relationships among sensor events refer to semantic relationship herein [3–7].


M. Maximilien et al. (Eds.): ICSOC 2017, LNCS 10601, pp. 191–205, 2017.
https://doi.org/10.1007/978-3-319-69035-3_13

In our previous work [1, 2], we studied a new kind of relationship among sensor events, called statistical correlation. We used Pearson coefficient to measure such relationship. Specifically, we tried to map physical sensors into a software-defined abstraction, called proactive data service. A proactive data service takes event streams derived from physical sensors or other services as inputs and transforms them into new streams based on user-defined operations. In [2], we also proposed a new abstraction, called *service hyperlink*, to encapsulate correlations between streams received and outputted by two data services. With service hyperlinks, a service can dynamically route an event to other services at runtime. In this way, the knowledge segment about how these sensors collaborate with each other can be depicted at the software layer.

In this paper, we further refine event correlation on when and how a type of event causes another type. Such event correlation can be easily transformed into a relationship between two IoT services. The main contributions include: (1) We propose an algorithm, called *CorFinder*, to discover such event correlations in a log of sensor events. To reach this goal, we update classic frequent sequence mining algorithm. (2) In a real application, we apply our approach to make anomaly warnings in a power plant based on discovered event correlations. We elaborate on how our approach works and what the differences with the traditional approaches are. (3) Furthermore, a lot of experiments are done to show the effectiveness of our approach based on a dataset from a power plant.

## 2 Problem Analysis

Figure 1 shows a real case of anomaly detection in a power plant. Fan stall is a major failure for the important equipment – primary air fan (PAF) in a power plant. It will cause severe damages to the whole air and flue system. Currently, detecting such equipment failures in a power plant mainly depends on the observation and judgment of envelope range. They detect anomalies through various phenomena, like the sharp descending of exit air pressure, electricity, and air volume in a PAF. However, when such phenomena are observed, an anomaly has already occurred and the loss is inevitable.

From a systematic view, a severe failure is often caused by some trivial anomalies step by step. The paths of anomaly propagation are usually hidden behind the correlations of sensor events in an IoT system. Figure 1 shows several possible event propagation paths lead to the fan stall failure. We can observe that each propagation path is formed of several correlated sensors.

For example, a decrease of valve degree (Valve Degree Descending Event) will reduce the inlet air header pressure (Inlet Air Header Pressure Descending Event). To maintain the output of the boiler, valve degree (Valve Degree Ascending Event) is automatically increased to prevent inlet air header pressure event from decreasing in this case. Following its rise, air pressure (Air Degree Ascending Event) increases and will lead to the growth of electricity and exit air pressure. Unfortunately, excess air pressure will cause a fan stall, which manifests as a sharp drop of electricity (Electricity Descending Event) and exit air pressure (Exit Air Pressure Descending Event).

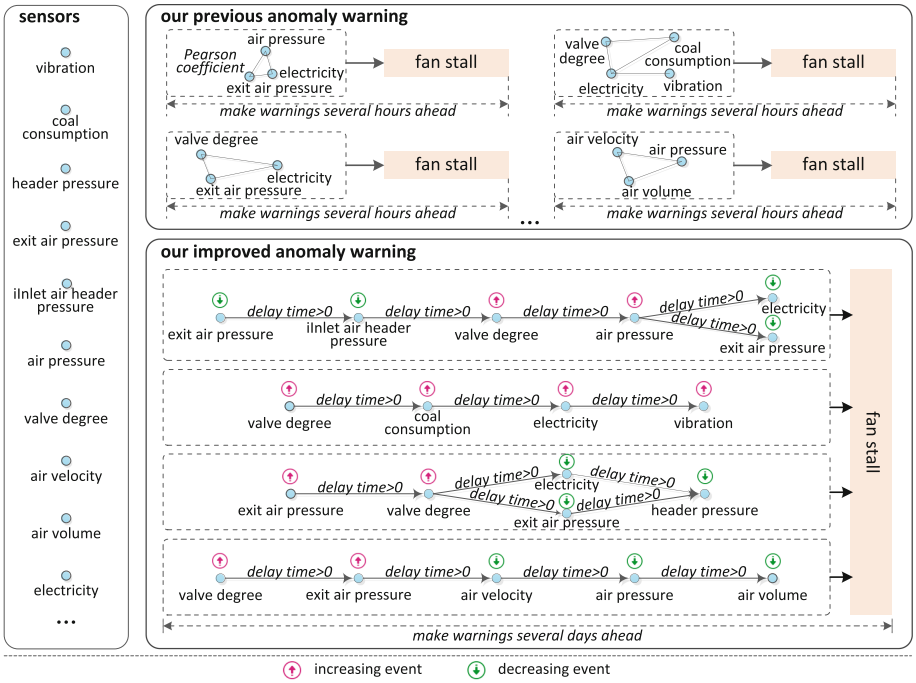


**Fig. 1.** Partial possible cases of fan stall in the primary air fan: a real case.

However, we find such correlations are not always available. For example, considering *exit air pressure* sensor and *inlet air header pressure* sensor, their correlations only exist when the value of *exit air pressure* sensor exceeds 5. In this situation, the value of *inlet air header pressure* sensor usually will keep the accordance with *exit air pressure* sensor after about 3 min. Lots of similar cases can be found.

The above case shows, to make warnings in advance, we need to clearly under-stand the way how an event transforms itself and propagates among different devices. An effective way is to mine the event correlations. If we find such correlations, we can merge these correlations to form an event propagation path as Fig. 1 shows.

## 3 Definitions

A sensor event $e$ consists of four elements: *a generation timestamp*, *a unique identifier, a sensor id* and *a value*. A sensor event log records events from all sensors in an IoT system. We formulate a sensor event log as follows.

**Definition 1** (Sensor Event log): given a set of sensors $S = \{s_1, s_2, \ldots, s_m\}$, a sensor event log is a set of sensor events $L = \{e_1, e_2, \ldots, e_n\}$, where $e_i (i = 1, .., n)$ is a sensor event generated from a sensor $s_j \in S$.

For example, a sample of sensor event log is $L$ = {
*2015-11-15 02:24:20, 118967, A110(Valve Degree), 0.359347557;*
*2015-11-15 02:24:20, 118968, A763(Coal Consumption), 36.54394756;*
*2015-11-15 02:24:20, 118969, A945(Electricity), 123.4148096;*
*2015-11-15 02:24:20, 118970, A658(Vibration), 97.32905983;*
*2015-11-15 02:24:21, 118967, A110(Valve Degree), 0.359347557; …*
}.

From a sensor event log $L$, an event sequence is a set of events $\mathcal{s} = \langle e_1, e_2, \ldots, e_k \rangle$ from the same sensor in ascending order by their timestamps. The correlation between event sequences is defined as follows.

**Definition 2** (Event Correlation): Given two event sequences $\mathcal{s}_i, \mathcal{s}_j \in L$, let $(\mathcal{s}_i, \mathcal{s}_j, \Delta t, conf)$ be the event correlation between $\mathcal{s}_i$ and $\mathcal{s}_j$, where $\mathcal{s}_i$ is the source, $\mathcal{s}_j$ is the target, $\Delta t$ is the time $\mathcal{s}_j$ delayed to $\mathcal{s}_i$, and *conf* is a measure of relationship between $\mathcal{s}_i$ and $\mathcal{s}_j$.

The left part of Fig. 2 elaborates an example of event correlation. In this picture, the red dashes line marks out $\mathcal{s}_i$ and $\mathcal{s}_j$ respectively. $\Delta t$ is 4 s.

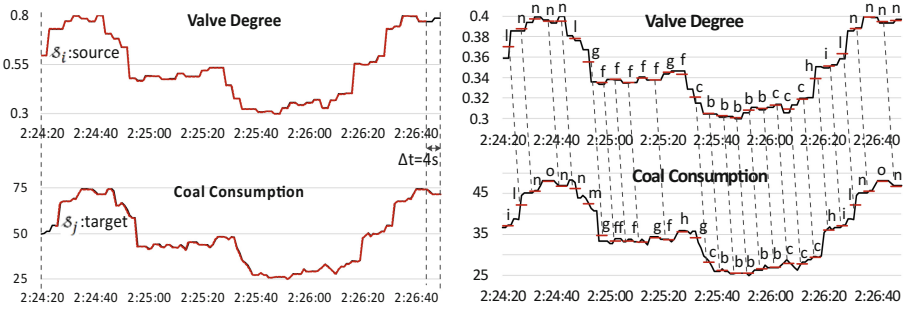


**Fig. 2.** An example of event correlation.

## 4 Discovery of Event Correlation

### 4.1 The Rationales

The main idea is to transform the event correlation discovery into a frequent sequence mining problem. To do this, as the right part of Fig. 2 shows, a numerical event sequence from a sensor is firstly transformed into a symbol sequence [8]. Essentially, symbolization is a coarse-grained description since each symbol corresponds to a segment of the original sequence. In this manner, if a sequence correlates with another one, there probably exists a frequent sequence between their symbolized sequences [8]. It inspires us to use the frequent sequence to measure event correlation. In another word, if two symbolized sequences $\mathcal{s}_i$ and $\mathcal{s}_j$ have a long enough frequent sequence, there is a correlation between them.

One challenge is how to identify the time delay between two correlated event sequences shown in Fig. 2. It actually reflects how long that a sensor will be affected by the value changes of its correlated sensor. However, traditional frequency sequence mining algorithm cannot directly solve such problem. Traditional algorithms only focused on the occurrence frequency of a sequence in a sequence set [9, 10]. Hence, we try to design an algorithm which can discover a frequent sequence, each element of which occurs in a sequence set within a short time period, i.e., time delay $\Delta t$ in Definition 2. Another challenge is how to determine the target and source by a frequent sequence. If each element of a frequent sequence occurs in same order in the sequence set, such frequent sequence can identify the target and source. Taking the right picture in Fig. 2 as an example, each element occurs a little earlier (no more than 4 s) in valve degree sequence than in coal consumption sequence. It indicates that valve degree sequence is the source, and coal consumption sequence is the target. In a word, if two symbol sequences $s_i$ and $s_j$ have a long-enough frequent sequence, each element of which occurs in $s_i$ and $s_j$ in same order within the time period $\Delta t$, the original sequences of $s_i$ and $s_j$ have an event correlation $(\mathcal{s}_i, \mathcal{s}_j, \Delta t, conf)$. In this way, *conf* can be computed as the ratio of the frequent sequence length to the length of $\mathcal{s}_i$.

Based on the above discussion, we propose an algorithm called *CorFinder* to discover event correlations. Firstly, it uses a classic algorithm, called *SAX* [8], to symbolize each event sequence in a sensor event log. Secondly, it mines the above frequent sequences. Notably, we take gap constraint [9] into consideration. A gap constraint $\gamma$ means any two adjacent elements in a frequent sequence skip no more than the predefined consecutive elements in any sequence containing the frequent sequence. A gap constraint can identify uncorrelated segments from correlated sequences.

**Symbolization.** In this paper, the classic symbolic representation algorithm Symbolic Aggregate approXimation (*SAX*) [8] is used to preprocess our input numerical event sequences. *SAX* algorithm allows an event sequence of length $n$ to be reduced to a symbol sequence of length $m$ ($m \ll n$) composed of $k$ different symbols. We will attach a timestamp to each symbol. The sequences in Table 1 are the symbolization of four event sequences from a sensor event log in a power plant via *SAX* algorithm with $k = 15$. The first two event sequences are shown in Fig. 2.

**Table 1.** A sample of a symbolized event sequence set (running example).

| *SID* | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ | $t_{18}$ | $t_{19}$ | $t_{20}$ | $t_{21}$ | $t_{22}$ | $t_{23}$ | $t_{24}$ | $t_{25}$ | $t_{26}$ | $t_{27}$ | $t_{28}$ | $t_{29}$ | $t_{30}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VD | *l* | *n* | *n* | *n* | *n* | *l* | *g* | *f* | *f* | *f* | *f* | *f* | *g* | *f* | *c* | *b* | *b* | *b* | *b* | *b* | *c* | *c* | *c* | *h* | *i* | *l* | *n* | *n* | *n* | *n* |
| CC | *i* | *l* | *n* | *o* | *n* | *n* | *m* | *g* | *f* | *f* | *f* | *g* | *f* | *h* | *g* | *c* | *b* | *b* | *b* | *b* | *b* | *c* | *c* | *c* | *h* | *i* | *l* | *n* | *o* | *n* |
| E | *i* | *k* | *m* | *n* | *o* | *n* | *n* | *l* | *e* | *e* | *e* | *e* | *e* | *c* | *e* | *c* | *b* | *b* | *b* | *b* | *b* | *c* | *e* | *f* | *g* | *i* | *k* | *m* | *n* | *o* |
| V | *i* | *j* | *n* | *o* | *o* | *n* | *n* | *k* | *e* | *f* | *f* | *f* | *f* | *f* | *g* | *d* | *b* | *b* | *b* | *b* | *b* | *c* | *c* | *d* | *f* | *i* | *j* | *n* | *o* | *o* |

VD: Valve Degree, CC: Coal Consumption, E: Electricity, V: Vibration; $t_1$ = 02:24:24, $t_2$ = 02:24:29, $t_3$ = 02:24:34, …, $t_{30}$ = 02:26:49.

**Frequent Sequence Mining.** Before introducing our algorithm, we list some related concepts about frequent sequence mining. A sequence in a sequence set $D$ is associated

with an identifier, called a *SID*. A support of a sequence is the number contained in $D$. A sequence becomes frequent if its support exceeds a pre-specified minimum support threshold in $D$. A frequent sequence with length $l$ is called $l$-frequent sequence. It becomes closed if there is no super-sequence of it with the same support in $D$. A projection database of sequence $s$ in $L$ is defined as $D_s = \{\alpha | \eta \in D, \eta = \beta \diamond \alpha\}$ ($\beta$ is the minimum prefix of $\eta$ containing $s$).

Projection-based algorithms are a classic category of traditional algorithms in frequent sequence mining [10]. They adopt a divide-and-conquer strategy to discover frequent sequences by building projection database. These algorithms firstly generate 1-frequent sequences $F_1$, where $F_1 = \{s_1 : sup_1, s_2 : sup_2, \ldots, s_n : sup_n\}$, $s_i$ is a 1-frequent sequence and $sup_i$ is its support. This step is followed by the construction of projection database for each 1-frequent sequence. In each projection database above, they generate 1-frequent sequences $F_2$ and projection database of each element in $F_2$. The process is repeated until there is no 1-frequent sequence. We propose two data structures as follows to update the classic algorithms.

**Loose $(\lambda, \Delta t, l)$-frequent sequence and $\lambda$-projection database.** We propose several concepts in this section. Traditionally, a frequent sequence with length 1 is called 1-frequent sequence. In this paper, a 1-frequent sequence occurring in time period $\Delta t$ is called $(\Delta t, 1)$-frequent sequence. The concept is extended as loose $(\Delta t, 1)$-frequent sequence $s' : \langle (SID_1, t_1), (SID_2, t_2), \ldots, (SID_m, t_m) \rangle$, where $s'$ occurs in $SID_i$ at $t_i$ and $t_{i+1} - t_i \leq \Delta t$. Generalize loose $(\Delta t, 1)$-frequent sequence into length $l$ as follows. Given a set of $(\Delta t, 1)$-frequent sequences $s'_1, s'_2, \ldots, s'_l$ for id-list $\langle SID_1, SID_2, \ldots, SID_m \rangle$, if $s'_1, s'_2, \ldots, s'_l$ orderly occurs in $SID_j (j = 1, 2, \ldots, m)$, $s' = \langle s'_1, s'_2, \ldots, s'_l \rangle$ is a loose $(\Delta t, l)$-frequent sequence for the id-list. A loose $(\Delta t, l)$-frequent sequence $s'$ becomes a loose $(\lambda, \Delta t, l)$-frequent sequence if it satisfies gap constraint $\lambda$, i.e., $s'_i$ and $s'_{i+1}$ $(i = 1, 2, \ldots, l-1)$ skips no more than $\lambda$ consecutive elements in $SID_j (j = 1, 2, \ldots, m)$.

According to previous analysis, loose $(\lambda, \Delta t, l)$-frequent sequences is the formulation of the frequent sequences our algorithm focuses on. It can identify our event correlations. To discover loose $(\lambda, \Delta t, l)$-frequent sequences, we propose $\gamma$-projection database. $\gamma$-projection database of sequence $s$ is denoted as $\gamma D_s = \{\alpha | \eta \in D, \eta = \beta \diamond \theta, \theta = \alpha \diamond \mu\}$ ($\beta$ is the minimum prefix of $\eta$ containing, $\alpha$ is the prefix of $\theta$ with length $\gamma + 1$).

Some examples of the above concepts are shown in Table 1. Let $\Delta t = 5$ s and $\gamma = 2$. $l$: $\langle (VD, t_1), (CC, t_2) \rangle$ is a $(\Delta t, 1)$-frequent sequence (grey squares in Table 1); $c$: $\langle (E, t_{14}), (VD, t_{15}), (CC, t_{16}) \rangle$ is a loose $(\Delta t, 1)$-frequent sequence (blue squares in Table 1), and $\{(VD, \langle (b,t_{16}), (b,t_{17}) \rangle), (CC, \langle (b,t_{17}), (b,t_{18}) \rangle), (E, \langle (e,t_{15}), (c,t_{16}) \rangle)\}$ is its $\gamma$-projection database (red squares in Table 1); $\langle c, h \rangle$: $\langle (VD, \langle t_{23}, t_{24} \rangle), (CC, \langle t_{24}, t_{25} \rangle) \rangle$ is a loose $(\gamma, \Delta t, 2)$-frequent sequence (green squares in Table 1); $\langle c, i \rangle$: $\langle (E, \langle t_{22}, t_{26} \rangle), (V, \langle t_{22}, t_{26} \rangle) \rangle$ is a loose $(\Delta t, 2)$-frequent sequence but not $(\gamma, \Delta t, 2)$ one (purple squares in Table 1).

### 4.2 The *CorFinder* Algorithm

In this paper, we improve the classic projection-based algorithms and propose the *CorFinder* algorithm to solve our problem. Traditional 1-frequent sequence *s*:*sup* does not consider occurrence time of *s*. Consequently, we propose the concept of $(\Delta t, 1)$-frequent sequence. However, any adjacent $(\Delta t, 1)$-frequent sequences for same sequence *s* are overlapped. It will increase storage cost and lead to repeated counting. For instance, adjacent $(\Delta t, 1)$-frequent sequences for *c*, $c$:$\langle(E, \mathrm{t}_{14}), (VD, \mathrm{t}_{15})\rangle$ and $c$: $\langle(VD, \mathrm{t}_{15}), (CC, \mathrm{t}_{16})\rangle$, are overlapped in $(VD, \mathrm{t}_{15})$. Therefore we extend $(\Delta t, 1)$-frequent sequence into loose $(\Delta t, 1)$-frequent sequence. The following Theorem 1 lays the foundation of the completeness of our algorithm.

**Theorem 1.** Each $(\Delta t, 1)$-frequent sequence in a given sequence set *D* is contained by a loose $(\Delta t, 1)$-frequent sequence in *D*. Versa, any element of each loose $(\Delta t, 1)$-frequent sequence in *D* is contained by a $(\Delta t, 1)$-frequent sequence.

Proof. We prove the theorem by reduction to absurdity. Let *D* be a sequence set, and there is a $(\Delta t, 1)$-frequent sequence $s : \langle(SID_1, t_1), (SID_2, t_2), \ldots, (SID_k, t_k)\rangle$ in *D*. Assume that there is no loose $(\Delta t, 1)$-frequent sequence containing *s*. Thus, any $SID_i \in s(i<k)$, $t_{i+1} - t_i > \Delta t$. Obviously, $t_k - t_1 > (k-1) * \Delta t$. Therefore $s : \langle(SID_1, t_1), (SID_2, t_2), \ldots, (SID_k, t_k)\rangle$ is not a $(\Delta t, 1)$-frequent sequence. It is a contradiction in the assumption.

On the other hand, assume that there is an element $(SID_i, t_i)$ of a loose $(\Delta t, 1)$-frequent sequence $s' : \langle(SID_1, t_1), (SID_2, t_2), \ldots, (SID_m, t_m)\rangle$, and $(SID_i, t_i)$ is contained by none of $(\Delta t, 1)$-frequent sequences in *D*. Let $SID_j$ be the nearest element to $SID_i$ under $SID_i \neq SID_j$. Since $(SID_i, t_i)$ is not contained by any $(\Delta t, 1)$-frequent sequence, $|t_i - t_j| > \Delta t$. It is in contradiction with the assumption that $s'$ is a loose $(\Delta t, 1)$-frequent sequence. So far, Theorem 1 is proved.

Loose $(\Delta t, l)$-frequent sequence can tell the target and source in an event correlation while considering time delay $\Delta t$ between the target and source. It is a measure of our event correlation. Our *CorFinder* algorithm aims at discovering loose $(\lambda, \Delta t, l)$-frequent sequences for finding event correlations. The Theorem 2 inspires us to discover a loose $(\lambda, \Delta t, l)$-frequent sequence in $\gamma$-projection database of its *l*-1 prefix.

**Theorem 2.** Any loose $(\lambda, \Delta t, l)$-frequent sequence $\mathcal{s}' = \mathcal{s}'_{l-1} \diamond s'_l$ can be discovered in the id-lists of $\mathcal{s}'_{l-1}$ and $s'_l$, where $\mathcal{s}'_{l-1}$ is the prefix of $\mathcal{s}'$ with length *l*-1 and $s'_l$ is a loose $(\Delta t, 1)$-frequent sequence in $\gamma$-projection database of $\mathcal{s}'_{l-1}$.

Proof. Obviously, $\mathcal{s}'_{l-1}$ is a loose $(\lambda, \Delta t, l-1)$-frequent sequence. Let $\gamma D_{\mathcal{s}'_{l-1}}$ be the $\gamma$-projection database of $\mathcal{s}'_{l-1}$. Because $\mathcal{s}'$ is a loose $(\lambda, \Delta t, l)$-frequent sequence, assume its id-list is $\langle SID_1, SID_2, \ldots, SID_m\rangle$, we get $\{SID_1, SID_2, \ldots, SID_m\} \in \gamma D_{\mathcal{s}'_{l-1}}$ and $s'_l$ must be a loose $(\Delta t, 1)$-frequent sequence for the id-list. Therefore, $s'_l$ is a loose $(\Delta t, 1)$-frequent sequence in $\gamma D_{\mathcal{s}'_{l-1}}$. Theorem 2 is proved.

Theorem 2 indicates that we can discover a loose $(\lambda, \Delta t, l)$-frequent sequence $\mathcal{s}'$ with *l*-1 prefix $\mathcal{s}'_{l-1}$ by the following steps. (1) Generate $\gamma D_{\mathcal{s}'_{l-1}}$ and all loose $(\Delta t, 1)$-frequent sequence in $\gamma D_{\mathcal{s}'_{l-1}}$. (2) For each loose $(\Delta t, 1)$-frequent sequence $s'_l$, discover

frequent sequences in id-lists of $s'_{l-1}$ and $s'_l$. (3) Generate loose $(\lambda, \Delta t, l)$-frequent sequences in the frequent sequences.

Consequently, the recursion of generating $\gamma$-projection databases and loose $(\Delta t, 1)$-frequent sequences can discover all loose $(\lambda, \Delta t, l)$-frequent sequences. Finally, *CorFinder* algorithm can discover event correlations by these loose $(\lambda, \Delta t, l)$-frequent sequences.

## 5 Application of Event Correlation for Anomaly Warning

### 5.1 The Service Collaboration Framework

Our previous work proposed an IoT service model to encapsulate sensor events into a service [1, 2]. It can serve as the fundamental unit to form an IoT application. When building a service, a user customizes its functionality by customizing the input event sensors as well as operations. Each service processes its input sensor events by predefined operations and generates higher-level events in form of stream. A created service can be encapsulated into a Restful-like API so that other services or applications can use it conveniently and simply. Moreover, our service has an important component, which is called service hyperlink. Hyperlink is responsible for indicating target services for an outputted event. In this way, our services can run proactively to

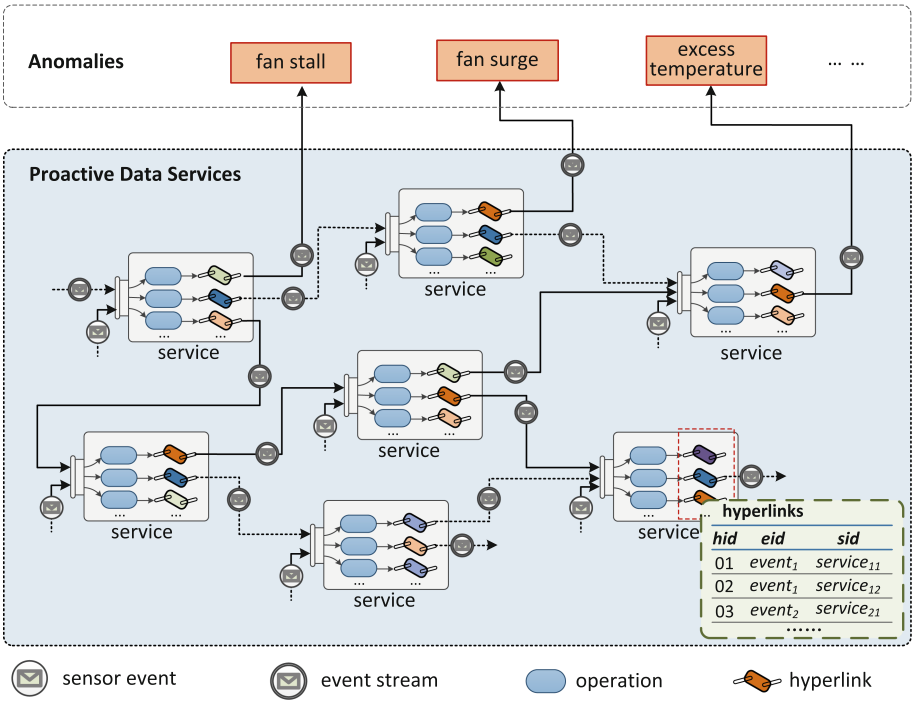


**Fig. 3.** The framework of our approach to correlating and collaborating with sensor events.

correlate and collaborate with sensor events to serve IoT applications. Figure 3 presents the framework of our approach.

Different from traditional service models and frameworks with the "request-and-response" model, ours works in a more automatic and real-time way with the 'stimuli-and-response' pattern while maintaining the common data service capabilities. To reach this goal, service hyperlink is the key point. A service hyperlink enables higher-level events outputted from a service (source service) to be routed to another one (target service). After a higher-level event is routed to a target service, the target service will be stimulated and autonomously respond to the event.

Our previous work encapsulated correlations among input sensor events as service hyperlinks and used Pearson coefficient to weigh the correlation degree. However, it is hard to tell the source and the target between two correlated services. To consummate the previous work, we encapsulate event correlation in this paper as service hyperlinks. With hyperlinks, a service can route an event to another service involving the target sequence of encapsulated event correlation.

### 5.2 The Process to Make Anomaly Warnings in a Power Plant

**Service Customization.** Making early warnings in a power plant is a typical case for our framework. As the beginning of the paper elaborates, we make early warnings by event propagation paths, e.g., a valve degree ascending event propagates along the way as valve degree → coal consumption → electricity → vibration and finally leads to a fan stall in Fig. 1. To reach this goal, we create services inputting sensor events from different sensors. Each service will detect and output trivial anomaly events, such as a valve degree ascending event. How to define and detect the trivial anomaly events precisely is the first problem in this case. It can be solved in two ways. On the one hand, such events can be defined based on business knowledge. On the other hand, those events can be identified by clustering techniques [11]. According to the defined events, we customize operations in each service so that a service can detect these trivial anomaly events autonomously.

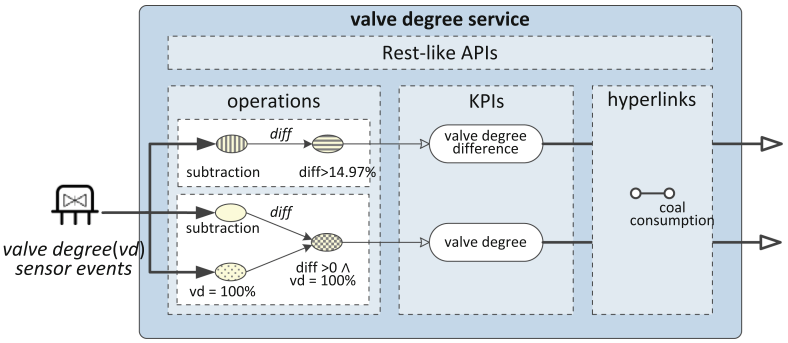


**Fig. 4.** The example of valve degree service.

For example, we build a valve degree data service as Fig. 4 presents. In this service, we select valve degree sensor events as its inputs. To detect a valve degree ascending event, we customize subtraction as one of its operations. The subtracting operation will subtract the value of a sensor event from that of the previous one. We perform K-means algorithm on a real data set within 6 months in a power plant and conclude that valve degree difference (short for *diff*) exceeding 14.97% is a trivial anomaly event. Thus, a filtering operation $diff > 14.97\%$ is selected to detect valve degree ascending events. Besides, inspection man concludes that the valve degree suddenly opening to all is a trivial anomaly event. According to the business knowledge, we select another filtering operation: $diff > 0 \wedge$ valve degree $= 100\%$. Valve degree and valve degree difference is the key attributes (KPIs) to be exposed with REST-like APIs. Based on the Fig. 2, the hyperlink of this service indicates that coal consumption service is its target service.

**Event Propagation.** A service hyperlink encapsulates an event correlation $(\mathcal{s}_i, \mathcal{s}_j, \Delta t, conf)$. An outputted event $e$ related to sensor of $\mathcal{s}_i$ will be routed along the hyperlink to its target service. The target service keeps detecting trivial anomaly events. If it detects $e$' with respect to sensor of $\mathcal{s}_j$ in time period $\Delta t$ after $e$ arrives, the target service will record a composite event by appending $e$ to trivial anomaly event $e$'. Instead of $e$', the composite event will be routed along the hyperlink related to $e$'. A composite event records the event propagation path. Figure 5 presents four correlated services. The composite event in vibration service indicates an event propagation path as valve degree ascending event → coal consumption ascending event → electricity ascending event → vibration ascending event.

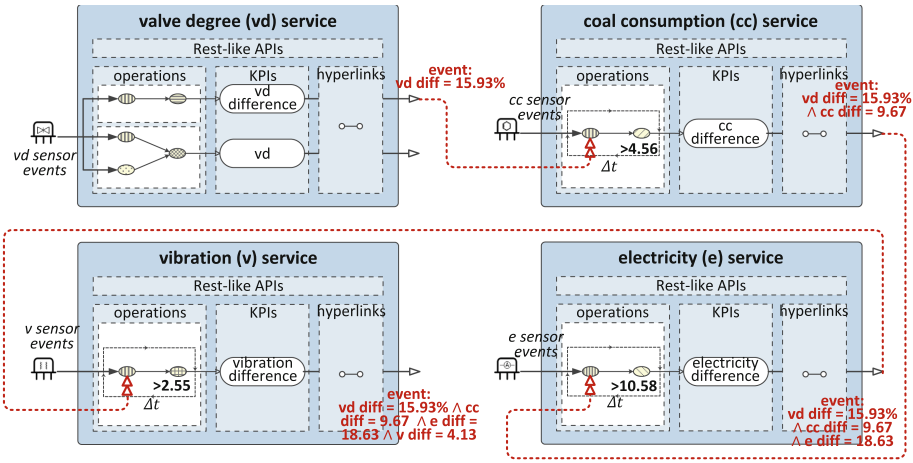


**Fig. 5.** An example of event propagation path.

**Anomaly Warning.** So far, we can get the propagation paths of trivial anomaly events in each service. But it is still insufficient to make early warnings since the trivial anomaly events are not equal to equipment anomalies. Practically, an inspector

performs scheduled maintenances and records equipment anomalies in maintenance records. A maintenance record $r = \langle rid, anomaly\_desc, rec\_time, anomaly\_obj \rangle$ consists of record id, anomaly description, recorded time, and anomaly object. For example, there is a maintenance record $r = \langle 118977,$ *vibration increases - fan stall*, 2015/10/12 05:12:00, *vibration in #2 primary air fan in #3 boiler*$\rangle$. According to recorded time and anomaly description in a maintenance record, we can infer causality between event propagation paths and anomalies. For instance, an event propagation path in Fig. 5 often occurred before a fan stall anomaly. Thus we can infer causality as valve degree ascending event $\rightarrow$ coal consumption ascending event $\rightarrow$ electricity ascending event $\rightarrow$ vibration ascending event $\Rightarrow$ fan stall. Once such a propagation path occurs in the runtime, a warning of a fan stall can be made. Consequently, each service is initialized with an operation for comparing runtime event propagation paths with historical ones. This operation takes composite events as input, and outputs warnings to users or other applications. The process to make anomaly warnings in a service after receiving a sensor event is shown in Fig. 6.

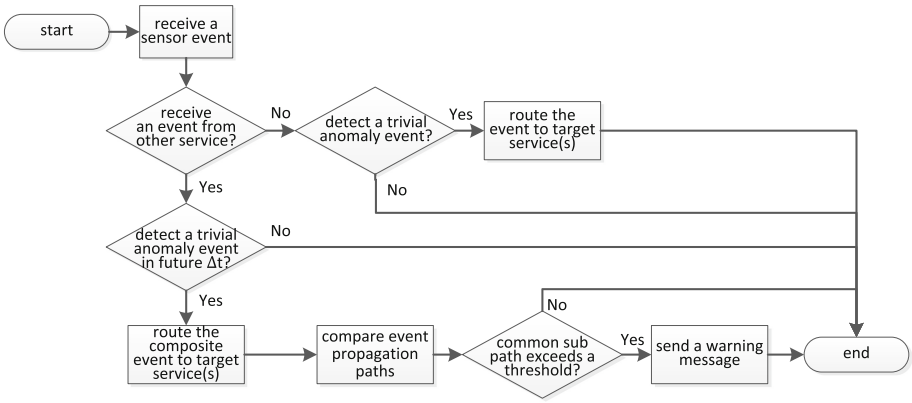


**Fig. 6.** Process of responding stimuli autonomously and proactively in a service.

## 6 Experiments

### 6.1 Experiment Setup

**Datasets:** The following experiments use a sensor event log from a power plant. The log contains sensor events from 2015-07-26 23:58:30 to 2016-08-17 07:55:00. Totally 480 sensors are involved and each sensor generates one event per second. The log is divided into two sets. The training set is from 2015-07-26 23:58:30 to 2016-01-31 23:59:55. This set is responsible for discovering event correlations. The testing set is from 2016-02-01 00:00:00 to 2016-08-17 07:55:00. It is used for making early warnings by our approach. In this set, events from same source are sent to our services as a stream. The time interval between two adjacent events is in accordance with real

intervals when they were generated. Besides, we use maintenance records of this plant power from 2015-07-26 23:58:30 to 2016-01-31 23:59:55 to verify the accuracy of our approach.

**Environments:** The experiments are done on a PC with four Intel Core i5-2400 CPUs 3.10 GHz and 4.00 GB RAM. The operating system is Windows 7 Ultimate. All the algorithms are implemented in Java with JDK 1.8.0.

### 6.2 Experiment Results

To verify the effectiveness of our approach, firstly, we create services according to physical sensors. We learn business knowledge from a power plant during the creation. Besides, sensors related to one attribute of devices' status are inputted into one service, such as events from *bearing temperature 1*, *2*, *3* and *4* sensor in primary air fan are the inputs of bearing temperature service. We created 108 services from all 440 sensors. Secondly, we input the training set into *CorFinder* algorithm to discover service hyperlinks. Next, on top of business knowledge and K-means clustering algorithm, we customize operations in our services to detect trivial anomaly events. After this, we sent testing set into our services as event streams. Once a service makes an early warning of an anomaly, it will print the message in the console. We compare the warnings with maintenance records to verify the accuracy of our approach. To measure the accuracy, we use the following indicators. **Precision** is the number of correct results divided by the number of all results. **Recall** is the number of correct results divided by the number of results that should have been returned. Notably, in this paper, our approach makes early warnings of the anomalies occurred both in training set and testing set.

To avoid loss, it is better to make early warnings of anomalies before they occur. To achieve this goal, we compute the precision and recall of our approach under different lengths of the trivial anomaly event propagation path. In the experiments, we set the length from 5 to 20 and draw the results as Figs. 7 and 8.

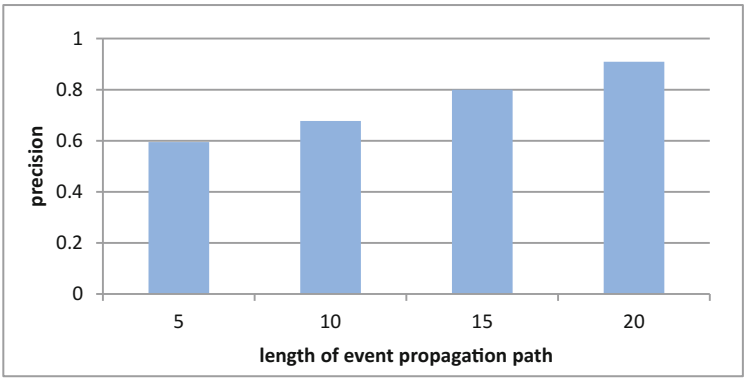


**Fig. 7.** The precision of our approach.

As Fig. 7 shows, the precision of our approach increases with the growth of propagation path's length. The reason is that longer propagation path can specify an anomaly more clearly. When the length is short, the event has multiple possible propagation paths so that it may evolve into different anomalies. Consequently, the shorter the length of event propagation path is, the lower the precision of our approach is. Meanwhile, shorter path needs less time to make a warning. It indicates that higher precision needs more time. In this experiment, our approach makes warnings of anomalies before the complete event propagation path is formed. It is the main reason that the precision keeps below 100%.

On the other hand, as the Fig. 8 shows, our approach's recall decreases with the rise of propagation path's length. Different from precision, our recall can reach 91.67% when the length is 5. It is because shorter event propagation path can specify more possibilities of anomalies, including those should have been made warnings. Besides, we analyze the details of the results and find that, regardless of the path's length, there are several anomalies our approach cannot discover. The reason is their propagation path is not completely covered by the paths in training set. Our approach cannot search the corresponding anomaly in training set. Fortunately, the anomaly occurs frequently in testing set, and we find that paths of the undiscovered anomalies can be covered by testing set. It inspires us to solve this problem by updating training set periodically.

Our experiment results show that we can make warnings of anomalies before they happen for 5 days ahead at most and 39.8 h ahead averagely, while the precision and recall exceeding 80%.

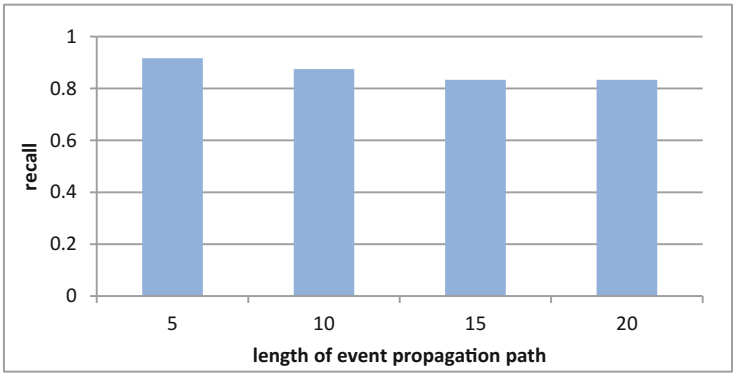


**Fig. 8.** The recall of our approach.

## 7 Related Works

Service correlation has attracted much attention in the field of service computing. Dong et al. tried to capture the temporal dependencies based on the amounts of calls to different services [12]. Hashmi et al. proposed a framework for web service negotiation management based on dependency modeling for different QoS parameters among multiple services [13]. Wang et al. considered that a dependency is a relation between

services wherein a change to one of the services implies a potential change to the others [14]. They utilized a service dependency matrix to solve the service replacement problem.

However, most of the existing work only considers input/output dependency, pre/post condition dependency, correlations among services and so on. Neither of them takes the dependency of the involved data, which can be regarded as events. Hence, existing studies of event correlation is also the foundation of our work.

Reguieg et al. regarded event correlation as correlation condition, which is a predicate over the attributes of events that can verify which sets of events belong to the same instance of a process [3]. It presented a framework and techniques with multi-pass algorithms to discover correlation conditions in process discovery and analysis tasks over big event datasets using MapReduce framework. It guarantees the efficiency and scalability by partitioning, replication and optimizing the I/O cost. Motahari-Nezhad et al. focused on event correlations in service-based processes [4]. It proposed the notion of correlation condition mentioned above. It developed an algorithm to discover event correlation (semi-) automatically from service interaction logs. Liu et al. presented an event correlation service for distributed middleware-based applications [5]. It enables complex event properties and dependencies to be explicitly expressed in correlation rules. Remarkably, these correlation rules can be accessed and updated at runtime. These event correlation studies provide foundations for our study. However, they do not consider the event correlation in an IoT environment.

Recently, some researchers focus on event dependencies. Song et al. mined activity dependencies (i.e., control dependency and data dependency) to discover process instances when event logs cannot meet the completeness criteria [6]. In this paper, the control dependency indicates the execution order and the data dependency indicates the input/output dependency in service dependency. A dependency graph is utilized to mine process instances. In fact, the authors do not consider the dependency among events. Plantevit et al. presented a new approach to mine temporal dependencies between streams of interval-based events. [7]. Two events have a temporal dependency if the intervals of one are repeatedly followed by the appearance of the intervals of the other, in a certain time delay.

## 8 Conclusion

In this paper, we elaborate service hyperlink by encapsulating event correlations in an IoT environment to consummate our previous work. We transform service hyperlink discovery into frequent sequence mining problem and propose the *CorFinder* algorithm. Moreover, we apply our approach to make anomaly warnings in a power plant. Experiments show that, our approach can make warning of anomalies before they happen for 5 days ahead at most, and 39.8 h ahead in average while the precision and recall exceed 80%.

**Acknowledgement.** Funding: This work was supported by National Natural Science Foundation of China (No. 61672042), Models and Methodology of Data Services Facilitating Dynamic Correlation of Big Stream Data; Beijing Natural Science Foundation (No. 4172018), Building Stream Data Services for Spatio-Temporal Pattern Discovery in Cloud Computing Environment; The Program for Youth Backbone Individual, supported by Beijing Municipal Party Committee Organization Department, Research of Instant Fusion of Multi-Source and Large-scale Sensor Data.

## References

1. Han, Y., Wang, G., Yu, J., Liu, C., Zhang, Z., Zhu, M.: A service-based approach to traffic sensor data integration and analysis to support community-wide green commute in china. IEEE Trans. Intell. Transp. Syst. **17**(9), 2648–2657 (2016)
2. Han, Y., Liu, C., Su, S., Zhu, M., Zhang, Z.: A proactive service model facilitating stream data fusion and correlation. Int. J. Web Serv. Res. **14**(3), 1–16 (2017)
3. Reguieg, H., Benatallah, B., Nezhad, H.R.M., Toumani, F.: Event correlation analytics: scaling process mining using mapreduce-aware event correlation discovery techniques. IEEE Trans. Serv. Comput. **8**(6), 847–860 (2015)
4. Motahari-Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event correlation for process discovery from web service interaction logs. VLDB J. **20**(3), 417–444 (2011)
5. Liu, Y., Gorton, I., Lee, V.: The architecture of an event correlation service for adaptive middleware-based applications. J. Syst. Softw. **81**(12), 2134–2145 (2008)
6. Song, W., Jacobsen, H.A., Ye, C., Ma, X.: Process discovery from dependence-complete event logs. IEEE Trans. Serv. Comput. **9**(5), 714–727 (2016)
7. Plantevit, M., Robardet, C., Scuturici, V.M.: Graph dependency construction based on interval-event dependencies detection in data streams. Intell. Data Anal. **20**(2), 223–256 (2016)
8. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 2–11. Association for Computing Machinery, San Diego, CA, United States (2003)
9. Pei, J., Han, J., Wang, W.: Constraint-based sequential pattern mining: the pattern-growth methods. J. Intell. Inf. Syst. **28**(2), 133–160 (2007)
10. Mooney, C.H., Roddick, J.F.: Sequential pattern mining - approaches and algorithms. ACM Comput. Surv. **45**(2), 1–39 (2013)
11. Ahmed, M., Mahmood, A.N., Islam, M.R.: A survey of anomaly detection techniques in financial domain. Future Gener. Comput. Syst. **55**(6), 278–288 (2016)
12. Dong, F., Wu, K., Srinivasan, V., Wang, J.: Copula analysis of latent dependency structure for collaborative auto-scaling of cloud services. In: Proceedings of the 25th International Conference on Computer Communication and Networks, pp. 1–8. Institute of Electrical and Electronics Engineers Inc., Waikoloa, HI, United States (2016)
13. Hashmi, K., Malik, Z., Najmi, E., Alhosban, A., Medjahed, B.: A web service negotiation management and QoS dependency modeling framework. ACM Trans. Manag. Inf. Syst. **7**(2), 1–33 (2016)
14. Wang, R., Peng, Q., Hu, X.: Software architecture construction and collaboration based on service dependency. In: Proceedings of 2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design, pp. 91–96. Institute of Electrical and Electronics Engineers Inc., Calabria, Italy (2015)