A DECENTRALIZED AND SERVICE-BASED APPROACH TO PROACTIVELY CORRELATING STREAM DATA

Yanbo Han[1,2], Chen Liu[1,2], Shen Su[1,2], Meiling Zhu[1,2,3], Zhongmei Zhang[1,2,3]
[1] Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data, North China University of Technology, Beijing, China
[2] Cloud Computing Research Center, North China University of Technology, Beijing, China
[3] School of Computer Science and Technology, Tianjin University, Tianjin, China
hanyanbo@ncut.edu.cn, liuchen@ncut.edu.cn, johnsuhit@gmail.com, meilingzhu2006@126.com, gloria_z@126.com

Abstract

Stream data from devices and sensors are considered a typical kind of big data. Though being promising, they are useful only when we can reasonably correlate and effectively use them. Herein, services come back to the spotlight. The position paper reports some of our efforts in promoting service-based integration and correlation of such stream data in a real setting – monitoring and optimized coordination of individual devices in a power plant.

Keywords: IoT Service, Proactive Data Service, Service Hyperlink, Sensor Data, Correlating Stream Data

1. Introduction Sensors and IoT in Industry: A Typical Scenario

IoT (Internet of Things) allows industry devices to be sensed and controlled remotely, resulting in better efficiency, accuracy and economic benefit. With the new wave of the forth industry revolution, the IoT-based integration of physical systems and computer systems is gaining momentum (https://en.wikipedia.org/wiki/Industry_4.0. Industry 4.0 Wikipedia.). The massive real-time IoT data may drive computing processes and services, and can derive a new horizon of industry automation with decentralized sensing, reasoning and response. We will examine a partial but typical scenario of such change with the example of anomaly detection in a power plant.

In a power plant, there are hundreds of machines running continuously and thousands of sensors deployed to monitor machine status at real-time. The status attributes correlate with each other in multiple ways, and indicate equipment status and anomalies. Fig.1 illustrates some deployed sensors and their possible correlations in a real power plant. In this paper, we advocate a decentralized and service-based approach to dynamically correlating the sensor data and generating higher-level events for systems and people. This poses many research challenges, some of which are discussed in the paper include service modeling of big stream data as well as data-driven and programmable correlation through service hyperlinks.

Let us take the primary air fan (PAF) unit as an example. As left side of Fig.2 shows, a PAF is equipped with 44 sensors, which continuously generate stream data, such as temperature and vibration of fan bearing. The right side of

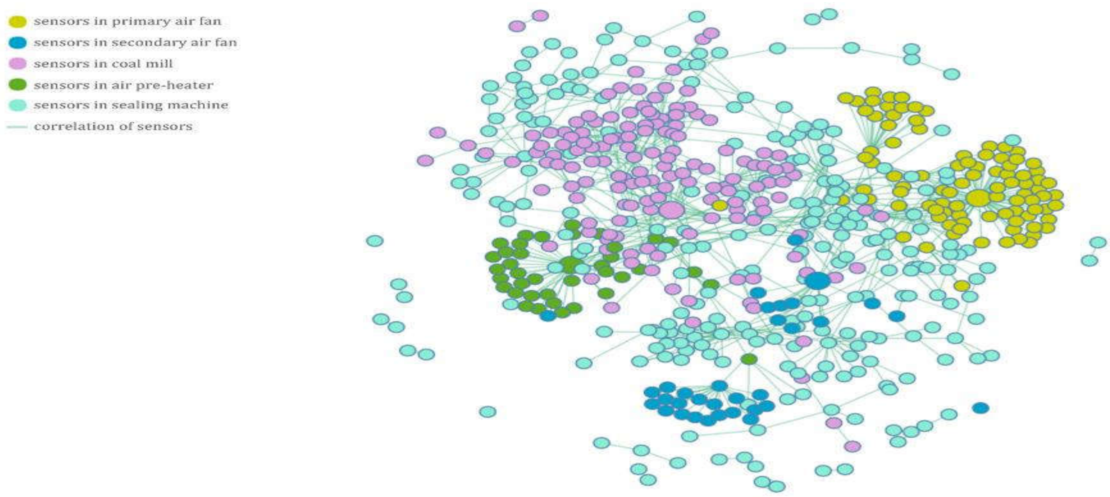

Figure 1. Snapshot of Partial Time-varying Correlations among Sensors in a Power Plant.

DOI: 10.29268/iciot.2016.0014

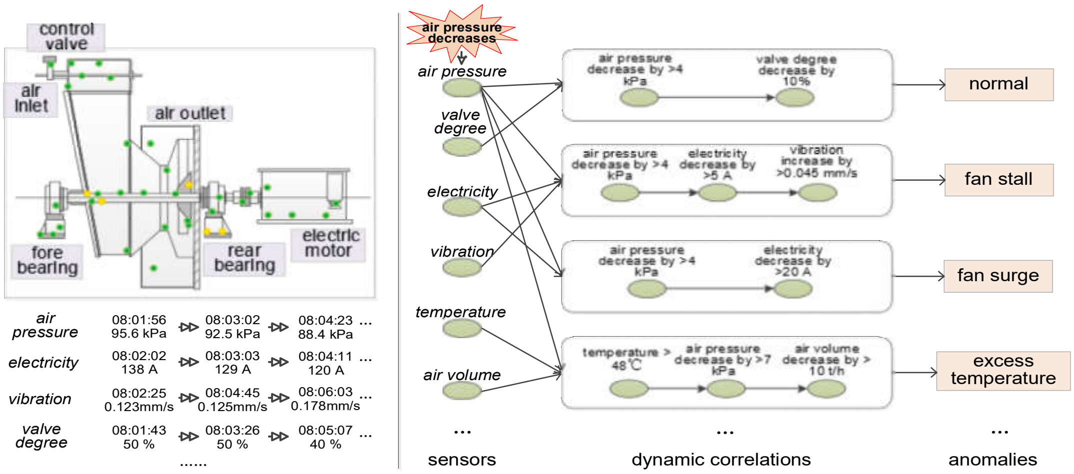

Figure 2. Possible Anomalies Led by the Quick Decrease of Air Pressure: a Real Case.

Fig.2 shows a partial process to detect anomalies in a PAF. The motivation is an observed event, which is the quick decrease of air pressure of a PAF. An air pressure decrease may cause fan stall, fan surge, or temporarily nothing. We cannot assure which result will be led at the beginning. It needs to make synthesized analysis with the other correlated sensor data and observe the data changes at runtime. For example, with the air pressure decreasing by 4kPa, the correlated sensor "degree of valve control" decreases by 10% may lead to the conclusion that the air pressure decrease is normal. But if the bearing vibration increases by over 0.045mm/s as well as the motor electricity decreases by over 5A next, there is a high possibility of the fan stall anomaly. Similarly, a fan surge can be confirmed if an air pressure decrease happens and the motor electricity decreases by over 20A. It is thus meaningful to correlate air pressure decrease and fan stall, and we need to check the bearing vibration and the motor electricity at runtime.

Note that we cannot just simply predefine a rule like "<air pressure decreases by 4kPa ∧ degree of valve control decreases by 10%> ⇒ the air pressure decrease is normal" to make the conclusion. The real case is much more complicated. Firstly, we cannot clearly give the boundary value for each rule. It is because the status of equipment and its environment keeps changing. Secondly, the runtime correlations among sensors vary along time. This leads to the multiple underlying correlations among sensors. Each one may lead to a kind of anomaly. Such path can only be clarified step by step at runtime following the process of situation changes and users' decisions. As a result, the above rules may not be enumerated and pre-defined at the beginning, and they should be formed at runtime and gradually consummated. Hence, we wish such fast-changing data could be automatically grouped together to generate higher-level events with service mechanisms. In fact, the dynamic correlations among status attributes are significantly prevalent. More efficient and effective anomaly detection can be achieved by dynamically correlating the sensor data.

2. Positioning of Services in the Context of Industry IoT

Business applications usually follow the "request-and-response" pattern. However, an IoT application can be more ad hoc and reactive. Industrial sensors generate successive on-site big stream data. It turns impractical for human beings to grasp the whole status with the traditional way of top-down programming and request-response thinking. Sporadic events are generated from the dynamic correlations of decentralized sensors along with the environment changes. For a user, it is hard to clarify the complex relationships or predict sporadic events.

Service-oriented paradigm has been seen as a mainstream approach for building large-scale distributed software systems. To decouple data to be shared from their sources, the concept of "data as service" or "data service" is proposed, which can provide semantically richer view and advanced querying functionality. The abstraction of sensor data services also gives us good opportunities to examine the way to build an IoT application. IoT applications possess some new intrinsic features that are different from other software applications. To deal with situational and ad-hoc problems, an IoT application should capture dynamic correlations of multiple sensors to respond more intelligently to various outside stimuli, like environment changes, sporadic events and so on.

Recently, efforts have been made to deal with big data by encapsulating common functionalities for storage, query, management, and analysis as services. For example, SenseWeb (Grosky, 2007) and Global Sensor Network (Aberer, 2007) provided platforms to assist users to share, manage, and access sensor data on-demand ubiquitously. Xu et al (Xu, 2014) and Perera et al (Perera, 2015) provided IoT data resources as services to support accessing cross-platform data by URI through Web for IoT applications. In our previous work (Han, 2015). we proposed a stream data service model to access stream data continuously in real-time, and implemented a carpooling service based on the stream data services. In practice, IoT applications always operate in highly dynamic environments. Some works (Potocnik, 2014; Bucchiarone, 2015; Cheng, 2016) concentrate on new types of services or incremental service composition methods to create situation-aware IoT applications. Potocnik et al defined a new service type — a Complex Event Aware (CEA) service that automatically reacts to complex events specified in its interface (Potocnik, 2014). And Cheng et al proposed an event-driven service coordination behavior model based on an extended event-condition-action (ECA) mechanism (Cheng, 2016).

We believe an IoT application should be built with the "stimuli-and-response" pattern. As shown in Fig 3, a novel type of service abstraction, called as ***proactive data service***, is proposed to serve as the fundamental unit to form an IoT application. Although lots of researches have focused on how to encapsulate sensor data into services, their traditional service model like REST service is still with the "request-and-response" model. With the proposed service model, we hope to find a more automatic and real-time way for handling sporadic events with the "stimuli-and-response" pattern while maintaining the common data service capabilities.

We blend an event model into our services. Each proactive data service can selectively respond to all events received from other services. There are multiple options to generate an event. For example, we can pre-setup a set of events, which could be caught or thrown by a data service. An event also can be generated by dynamically correlating various sensors. Especially when considering situation changes, correlations among different sensors can be regarded as important sources to generate underlying situational events.

Correlations among data services influence event routing. When an event routes from a source service to a target service, the target service will be stimulated to behave autonomous to respond to that event. Through this way, with an event spreading over, data services on its routing path are essentially composed.

3. CORRELATING STREAM DATA WITH PROACTIVE DATA SERVICES

3.1 Proactive Data Services

Before defining the Proactive Data Service model for stream data, we first state some related concepts. An event can be denoted by e. Every event is associated with a corresponding event source, attributes with corresponding values, and occurrence time. A particular event can be classified as a sensor event or a service event. The sensor event is generated directly by sensors and the service event is generated by services.

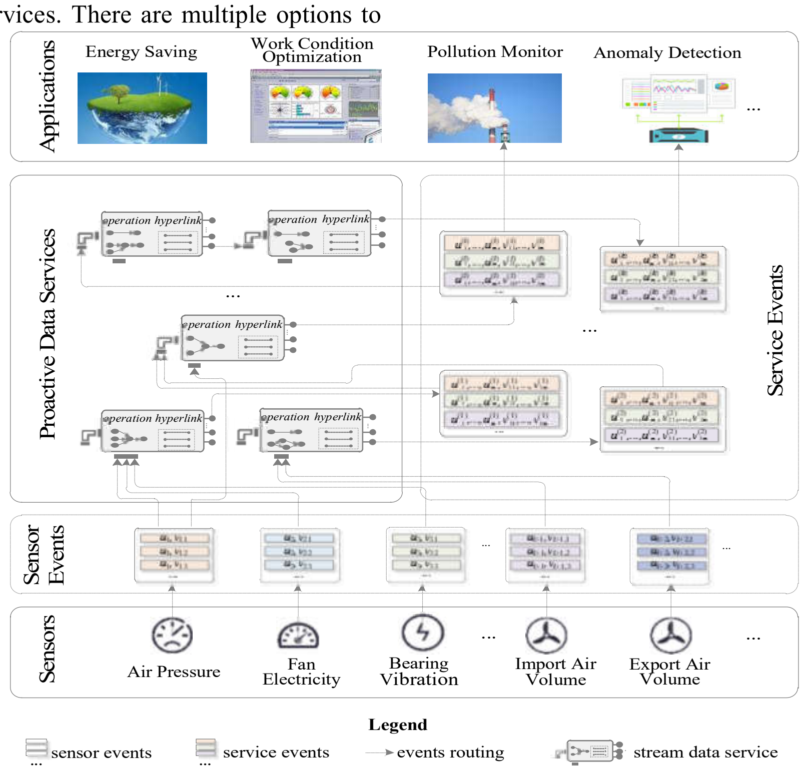

Figure 3. Rationale of Our Approach.

Definition 1. (Sensor Event): a sensor event can be represented as $e_f = \langle sid, p, t \rangle$, in which *sid* is the unique identifier of the sensor which generated e_f, $p = \langle a, v \rangle$ is a key-value pair, in which *a* means attribute and *v* means value, and *t* means the timestamp when the event occurs.

A service event is generated by transformation of multiple sensor events or the other service events.

Definition 2. (Service Event): a service event can be represented as $e_s = \langle sid, E, P, t \rangle$, in which *sid* is the unique identifier of the service generate e_s, $E = \{e_1, e_2, \ldots, e_n\}$, $n>0$, which is an event set that include several sensor events or service events which collectively constitute event e_s, $P=\{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \ldots \langle a_j, v_j\rangle\}$ is the concrete content generate by *E* which is a set of key-value pairs, and t means the timestamp when the service event is generated.

Correspondingly, a sensor can generate an event stream $\vec{E}$:

$$\vec{E} = \langle e_{f1}, e_{f2}, \ldots, e_{fi}, \ldots \rangle$$

In which, each sensor event has the same *sid* and the same attribute, and each sensor event disappears when new event appears by default. And service events with the same *sid*, same event set *E*, and same attribute sets can also form an event stream $\vec{E}$:

$$\vec{E} = \langle e_{s1}, e_{s2}, \ldots, e_{si}, \ldots \rangle.$$

In particular, we note an event *e* in event stream $\vec{E}_3$ $\vec{E}$ as $\vec{E}[t]$, in which *t* is the timestamp when *e* occurs, and $\vec{E}[t].a_i$ refers to *e*'s value of attribute a_i.

Traditional data service is software components that provide rich metadata and APIs for service consumers to send data requirements and receive data from service providers. Data service is a specialization of Web service which can be deployed on top of data stores, other services, or applications (Carey, 2012). However, because of the "request-response" model for traditional data service, it suffers certain limitations in an IoT environment, such as to collect, process, deliver and correlate continuous sensor data. To address these limitations, we define our proactive data service model based on the above definition of event and operation.

Definition3. (Proactive Data Service): We define a proactive data service as a 6-tuple as following (shown in Figure 4):

$S = \langle uri, in\_events, out\_events, operations, filter, hyperLinks \rangle$.

In which, *uri* is the unique identifier, *in_events* represents the input event channel receiving all event streams which arrive at the service, *filter* is responsible for deciding how to operate the received event streams and *operations* contains the corresponding operations, *out_events* represent the output event streams generated by *operations*, and *hyperLinks* refer to a routing table which is composed with multiple routing paths, and directs output events to the target services.

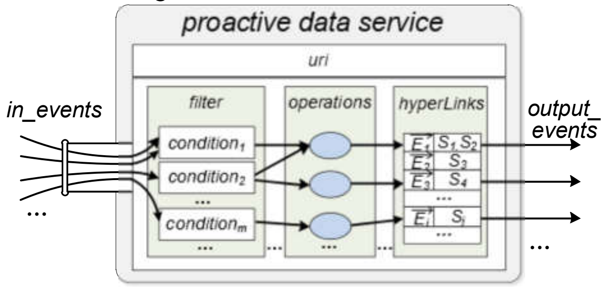

Figure 4. The Structure of Our Proactive Data Service.

Specifically, each operation in *operations* can be represented as $op_{trans} = \langle \vec{E}_{in}, func, \vec{E}_{out} \rangle$, in which, $\vec{E}_{in}$ is a set of input event stream, *func* is the transformation function, and $\vec{E}_{out}$ is an output event stream, in which each event is a service event.

We present some transformation operations refer to (Wang, 2016). Table I shows part of frequently used operations. Sensor and service events are both denoted by *e*.

Table I. Event Processing Operations.

Function	Expression	Description
CON(∧)	$e_1 \wedge e_2$	Conjunction of e_1 and e_2 without occurrence order
DIS(∨)	$e_1 \vee e_2$	Disjunction of e_1 and e_2 without occurrence order
NEG(~)	$\sim e_1$	Negation of e_1
SEL	*SEL*(e_1)	Select an event form input events
ANY(∃)	∃(e_1)	Any event that occurs of e_1 and e_2
SEQ	*SEQ*(e_1)	Select a given sequence of events from input events
EVERY(∀)	∀(e_1)	Every occurrence of e_1
AVE	*AVE*(a_1, e_1, e_2)	Average value of a_1 in e_1 and e_2
SUM	*SUM*(a_1, e_1, e_2)	Summation value of a_1 in e_1 and e_2
DIF	*DIF*(a_1, e_1, e_2)	Difference value of a_1 in e_1 and e_2
EQ	*EQ*(a_1, e_1, e_2)	Judge the equality of a_1 in e_1 and e_2
COUNT	*COUNT*(e_1)	Occurrence number of e_1
MAX	*MAX*(a_1, e_1, e_2)	Maximal value of a_1 in e_1 and e_2
MIN	*MIN*(a_1, e_1, e_2)	Minimum value of a_1 in e_1 and e_2
FIRST	*FIRST*(e_1, e_2)	First event of e_1 and e_2
LAST	*LAST*(e_1, e_2)	Last event of e_1 and e_2
WITHIN	*WITHIN*(e_1, t_1, t_2)	e_1 occurs within time intervals t_1 and t_2
WITHIN	*WITHIN*(e_1, t)	e_1 occurs within less than t
DURING	*DURING*(e_1, e_2)	e_1 occurs during e_2
WINDOW	*WINDOW*(e_1, t)	e_1 occurs for time period t
WINDOW	*WINDOW*(e_1, n)	e_1 occurs n times (n>0)
AT	*AT*(e_1, t)	e_1 occurs at time t

The *filter* and *hyperLinks* in our proactive service are the key distinctions compared to the traditional data service model. In particular, *filter* stores a set of rules, noted as conditions, to process the received events. Each type of input event $\vec{E_i}$ has a corresponding condition to determine which operations event *e* (i.e. $\vec{E_i}(t)$) in $\vec{E_i}$ will be sent to. A condition can be formalized as:

on event streams $\vec{E_i}$
if *constraint*$_1$ is satisfied
then invoke *operation 1*
if *constraint*$_2$ is satisfied
then invoke *operation 2*
… …

A condition includes *n* constraints (*constraint*$_1$, *constraint*$_2$, … , *constraint*$_n$), in which *constraint*$_i$ indicates the constraints (Li, 2009) on the content and timestamp of the context received events of the service. If *constraint*$_j$ is satisfied, the *filter* will invoke the corresponding *operation j* with the received events timestamp and content.

After processing with the operation, we encapsulate the output event with the result of the operation, current time, and predefined event attribute. Then send the generated output event to the *hyperLinks*. *hyperLinks* refer to a routing table which is composed with multiple routing paths, and directs output events to the target services. Each routing path $\langle \vec{E_i}, S_i \rangle$ indicates the target service when send the event stream $\vec{E_i}$. We will define *hyperLinks* after we define correlation.

3.2 Service-based Data Correlation and Hyperlinks

In the field of statistics, data correlation means the relationship between multi stochastic variables. Statists try to analyze and understand the data correlations. Herein, each series of sensor events or service events can be regarded as samples of a variable. Thus, to understand the correlation among them, we can learn from statistical correlation.

Furthermore, according to the definition proposed previously, each service may have multi event streams. To measure correlation between services, we define the correlation among event streams.

Definition 4. (Event Correlation): Given two sensor or service event streams $\vec{E_i}, \vec{E_j}$, we define the correlation between event stream $\vec{E_i}$ and $\vec{E_j}$ in time range *T* as the following matrix:

$$Cor(\vec{E_i},\vec{E_j},T) = \begin{bmatrix} cor(\vec{E_u}.a_{i1}, \vec{E_v}.a_{j1}, T) & \cdots & cor(\vec{E_u}.a_{i1},\vec{E_v}.a_{jn},T) \\ \vdots & \ddots & \vdots \\ cor(\vec{E_u}.a_{im}, \vec{E_v}.a_{j1},T) & \cdots & cor(\vec{E_u}.a_{im}, \vec{E_v}.a_{jn}, T)\end{bmatrix},$$

where $\vec{E_i}.a_{ip}$ (p = 1, 2, … , m) is an attribute in event stream $\vec{E_i}$ and $\vec{E_j}.a_{jq}$ (q = 1, 2, … , n) is an attribute in event stream $\vec{E_j}$, and $cor(\vec{E_u}.a_{ip}, \vec{E_v}.a_{iq}, T)$ is the correlation degree between the attribute $\vec{E_i}.a_{ip}$ and $\vec{E_j}.a_{jq}$ in time range *T*.

There are many classical measures of correlation degree among variables such as covariance matrix, Pearson correlation coefficient, longest common subsequence (LCSS), and probability.

As mentioned above, service hyperlinks are a collection of routing tables of a service. We collect them by the event correlations. We present the formal definition of service hyperlink on top of the previous concepts.

Definition 5. (Service Hyperlink) Given a service S_i, we define the hyperlinks of service S_i as the set of event correlations with source service S_i. Formally,

$$hyperLinks(S_i) = \{\langle \vec{E_u} \mid S_j \rangle \mid Cor(\vec{E_u}, \vec{E_v}, T) \geq \delta_{min}\}$$

where $\vec{E_u}$ is an event stream in S_i, $\vec{E_v}$ is an event stream in a target service S_j, $Cor(\vec{E_u}, \vec{E_v}, T)$ refers to the event correlation between $\vec{E_u}$ and $\vec{E_v}$ in a time range *T*, and δ_{min} refers to the minimum value of the correlation value.

As following is an example of hyperlinks, service S_1 has two output event streams $\vec{E_1}$ and $\vec{E_2}$; service S_2 has one output event stream $\vec{E_3}$; and service S_3 has one output event stream $\vec{E_4}$ Service S_1 continuously studies the correlation between $\vec{E_1}$, $\vec{E_2}$ and $\vec{E_3}$, $\vec{E_4}$, and find that $\vec{E_1}$ correlates with $\vec{E_3}$, and $\vec{E_2}$ correlates with $\vec{E_4}$. Thus the hyperlinks of service S_1 is $\{\langle \vec{E_1} \mid S_2 \rangle, \langle \vec{E_2} \mid S_3 \rangle\}$.

3.3 Quick and Dynamic Response with Proactive Data Services

Our service model aims at responding to the incoming events with runtime consideration. Receiving an event with certain content, our service model may take totally different reactions and operations in case of various environment, which correlates the other events already arrived at the service or on the way, so that each service can make its own decision in a bottom-up way, but not with a global intelligence.

When building a service, we first design the operations, input events, and output events of the service based on its aimed functionality. Then we design the filter logic by filling it up with conditions. The conditions dynamically react to each arrived event according to the events' content, filtering undesired events, and handle the desired events to corresponding operations. Next, we train each service's hyperlinks by computing the correlations between their output event streams and other service's output event streams within a sliding window. If the output event stream correlates to another service's output event stream, we save a routing path in the *hyperLinks* to direct that output event stream. As the event stream flushing in the sliding window, we need to recalculate and update the correlation between services.

Our claim of our service model's dynamic event correlation is twofold. First, we calculate the *hyperLinks* at runtime, which route the output events to the target service, thus the routing of the events among services is dynamic. Second, our service model's filter is composed with multiple conditions, whose reaction dynamically considers the other arrived events. As a result, receiving an input event, our service model may generate various reactions at runtime.

4. DISCUSSION WITH THE RUNNING SCENARIO

We now discuss the use of our service model in the scenario we introduced in section 1. Our data set includes the data collected from the 44 sensors deployed on the PAF, during the period from 2014-07-01 to 2016-01-31. Generally, we collect 278400 data records and observe 28 PAF anomalies including 15 fan stalls.

To detect anomalies of PAF based on our model at runtime, the first step is to extract data correlations among sensor data and the PAF anomalies. In fact, we cannot generate an overall idea of all correlations of the PAF problems. But connecting pieces of extracted correlations can also lead us to a non-trivial part of anomalies. This is the basic idea which our service model comes from. With the growing amount of sensors deployed, and the growing knowledge on the PAF problem by digging on the runtime and historical data, we can extract more correlations and detect more anomalies with confidence at runtime.

Now let's come back to our scenario, we first build up all necessary services to run the scenario. Table II shows the basic information of the proactive data services used in our scenario, including the name of the service, the operations, and the type of the output events. The Anomaly Alert service (S_1) is responsible for generating the logic of anomaly detection, and sends alert to corresponding maintenance staff when an anomaly happens. The Air Pressure Sensor service (S_5), the Vibration Sensor service (S_6), and the Electricity Sensor service (S_7) collect sensor data, conducting no operation, and transform the collected data into sensor events. The Air Pressure Decrease Anomaly service (S_2), the Vibration Increase Anomaly service (S_3), and the Electricity Decrease Anomaly service (S_4) catch the sensor events, filtering no events, and conduct corresponding operations to generate service events to feed our anomaly detection service. All events in our service model has one single attribute as described in Table II, e.g. $\vec{E_3}$'s event attribute is decrease of air pressure.

S_1 conducts two operations: 1) when it receives an event of air pressure decreases by over 4 kPa, and receives both vibration increases by over 0.045 mm/s and electricity decreases by over 5 A within a time window, S_1 generate a fan stall alert; 2) when it receives an event of air pressure decreases by over 4 kPa, and receives both event of electricity decreases by over 20 A within a time window, S_1 generate a fan surge alert. S_2, S_3, and S_4's operation calculate the value decrease/increase of the input events individually.

Next, we fill the service *filters* with conditions. Services S_5, S_6, and S_7 need no conditions in their filters. For services S_2, S_3, and S_4, they all have one single operation, and all input events are necessary for value change monitoring, thus their conditions send all input events directly to their corresponding operation. For service S_1, as mentioned above, three type of input events are considered to be processed: $\vec{E_3}$, $\vec{E_4}$, and $\vec{E_5}$. As shown in equation (1), (2), and (3), we used 3 conditions for S_1 to process the input events. Since an air pressure decreases by over 4 kPa may cause fan stall or fan surge, we invoke both operations of S_1 as shown in equation (1) with all such air pressure decrease events. Since an increase of vibration by over 0.045mm/s in case of the air pressure decrease over 4 kPa may increase our

Table II. The Detail of Services in Our Scenario.

ID	Name	Operations	Output Events
S_1	Anomaly Alert service	$\vec{E_3}$ [illegible] WITHIN [illegible] $\vec{E_4}$ [illegible], $\vec{E_3}$ [illegible], $\vec{E_3}$ [illegible] WITHIN [illegible] $\vec{E_5}$ [illegible], $\vec{E_3}$ [illegible], $\vec{E_3}$ [illegible] (1) $\vec{E_3}$ [illegible] WITHIN [illegible] $\vec{E_5}$ [illegible], $\vec{E_3}$ [illegible], $\vec{E_3}$ [illegible] (2)	fan stall ($\vec{E_1}$); fan surge ($\vec{E_2}$)
S_2	Air Pressure Decrease Anomaly service	*DIF* [illegible] *air pressure*, $\vec{E_6}$ [illegible], $\vec{E_6}$ [illegible]	decrease of air pressure ($\vec{E_3}$)
S_3	Vibration Increase Anomaly service	*DIF* [illegible] *vibration*, $\vec{E_7}$ [illegible], $\vec{E_7}$ [illegible]	increase of vibration ($\vec{E_4}$)
S_4	Electricity Decrease Anomaly service	*DIF* [illegible] *electricity*, $\vec{E_8}$ [illegible], $\vec{E_8}$ [illegible]	decrease of electricity ($\vec{E_5}$);
S_5	Air Pressure Sensor service		air pressure ($\vec{E_6}$)
S_6	Vibration Sensor service		vibration ($\vec{E_7}$)
S_7	Electricity Sensor service		electricity ($\vec{E_8}$)

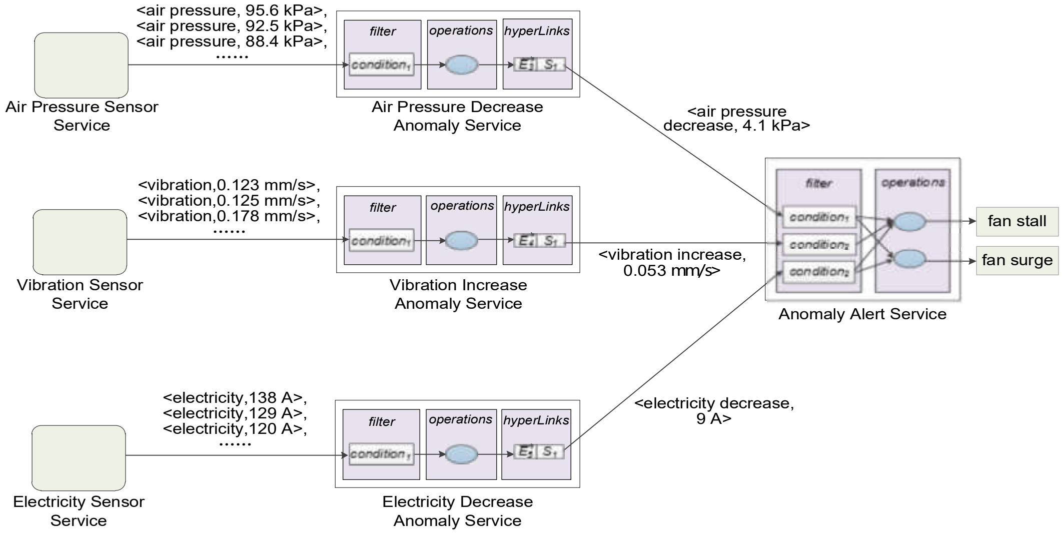

Figure 5. The Service Correlation Network in Our Scenario.

invoke operation (1) of S_1 (shown in table II) with such vibration increase events. We conduct similar condition design for the decrease of electricity to form condition 3.

on decrease of air pressure ($\vec{E_3}$)

if $\vec{E_3}.air\_pressure\_decrease > 4kPa$

then invoke *operation* (1) and *operation* (2)

on increase of vibration ($\vec{E_4}$)

if $\vec{E_4}.vibration\_increase > 0.045mm/s \wedge \vec{E_3}.air\_pressure\_decrease > 4kP$

then invoke *operation* (1)

on decrease of electricity ($\vec{E_5}$)

if $\vec{E_5}.electricity\_decrease < 5A \wedge \vec{E_3}.air\_pressure\_decrease > 4kPa$

then invoke *operation* (1)

if $\vec{E_5}.electricity\_decrease > 20A \wedge \vec{E_3}.air\_pressure\_decrease > 4kPa$

then invoke *operation* (2)

Next, let us connect the built services together, and tell them when and where to send each event by filling their *hyperLinks*. Initially, with no events running in our scenario, we calculate the output event correlations based on the historical data, and generate routing paths in the *hyperLinks* according to the correlations. Since fan stall and fan surge correlates to $\vec{E_3}$, $\vec{E_4}$, and $\vec{E_5}$, we save a routing path $\langle \vec{E_3}, S_1 \rangle / \langle \vec{E_4}, S_1 \rangle / \langle \vec{E_5}, S_1 \rangle$ in $S_2/S_3/S_4$'s *hyperLinks*. Since $\vec{E_3}$ correlates to $\vec{E_6}$, we save a routing path $\langle \vec{E_6}, S_2 \rangle$ in S_5's *hyperLinks*. Similar for hyperlinks of S_6 and S_7. Formally, S_2's *hyperLinks* could be represented as $\{\langle \vec{E_3}, S_1 \rangle\}$, similar for the other services.

Now the PAF anomaly detection service scenario is done. Let us take a runtime example with which generates a fan stall alert in steps:

1) S_5 keeps monitoring the air pressure, and sends the runtime event flow to S_2. As shown is Figure 5, S_5 sends 3 events to S_2 with the content key-value pairs of <air pressure, 95.6kPa>, <air pressure, 92.5kPa>, <air pressure, 88.4kPa>.
2) S_2 calculates the air pressure decrease according to the received events from S_5, and send them to S_1. In our scenario, S_2 sends 2 events e_1 e_2 with the key-value pairs of <air pressure decrease, 3.1kPa>, <air pressure decrease, 4.1kPa> to S_1.
3) S_1's *filter* handles the received events with condition (1). The event e_2 indicates the air pressure decreases by more than 4 kPa, according to condition (1), S_1invokes both operations of S_1.
4) Neither of S_1's two operations can generate a quick response to e_1's invoke, since both operations expect other input events. Thus S_1 holds e_1's timestamp and wait for the other input events.
5) S_6 generates the events of <vibration, 0.125mm/s>, <vibration, 0.178mm/s> and sends them to S_3. S_7 generates the event of <electricity, 138A>, <electricity, 129A> and sends them to S_4.
6) S_3 generates the event of e_3 <vibration increase, 0.053mm/s>, and sends it to S_1, invoking operation (1). S_4 generates the event of e_4 <electricity decrease, 9A>, and sends it to S_1, invoking operation (1). As a result, operation (1) generates a positive result. Since e_4 does not pass condition (3)'s checking constrains, operation (2) generates a negative result.
7) S_1 encapsulates the positive result of operation (1), and forms a fan stall alert.

5. SUMMARY

Data-driven approaches supporting locality and stimuli-response thinking are gaining momentum. This position paper presents our efforts in exploiting such possibilities on the basis of data service mechanisms. A novel service model for transforming and correlating massive stream data is proposed. This service model shows potential in realizing various middle-way programmable nodes to form larger-granularity software-defined 'sensors' in an IoT context.

6. REFERENCES

Grosky W I, Kansal A, Nath S, et al. (2007). SenseWeb: An infrastructure for shared sensing[J]. IEEE Multimedia, 14(4):8-13.

Aberer K, Hauswirth M, Salehi A. (2007). Infrastructure for data processing in large-scale interconnected sensor networks. International Conference on Mobile Data Management (MDM), 2007:198-205.

Xu B, Xu L D, Cai H, et al. (2014). Ubiquitous data accessing method in IoT-based information system for emergency medical services[J]. IEEE Transactions on Industrial Informatics, 10(2):1578-1586.

Perera C, Talagala D S, Liu C H, et al. (2015). Energy-efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in IoT clouds [J]. IEEE Transactions on Computational Social Systems, 2(4):171-181.

Han Y, Wang G, Yu J, et al. (2015). A service-based approach to traffic sensor data integration and analysis to support community-wide green commute in China. IEEE Transactions on Intelligent Transportation Systems, 1(9): 2648-2657.

Potocnik M, Juric M B. (2014). Towards complex event aware services as part of SOA[J]. IEEE Transactions on Services Computing, 7(3): 486-500.

Bucchiarone A, Sanctis M D, Marconi A, et al. (2015). Design for adaptation of distributed service-based systems. Service-Oriented Computing: International Conference (ICSOC), 2015: 383-393.

Cheng B, Zhu D, Zhao S, et al. (2016). Situation-aware IoT service coordination using the event-driven SOA paradigm[J]. IEEE Transactions on Network & Service Management, 13(2): 349-361.

Carey M J, Onose N, Petropoulos M. (2012). Data services[J]. Communications of the ACM, 55(6):86-97.

Wang D, Zhou M, Sajid A, et al. (2016). A Novel Complex Event Processing Engine for Intelligent Data Analysis in Integrated Information Systems[J]. International Journal of Distributed Sensor Networks, 2016(2):1-14.

Li X, Cheng B, Yang G, Liu Q. (2009). Method of Web Services Composition Based on Events[J]. Journal of Software, 20(12): 3101–3116.

Authors

Yanbo Han holds a PhD in computer science from the Technical University of Berlin in Germany. Dr. Han has been a full professor in computer science since 2000 (first with the Institute of Computing Technology, Chinese Academy of Sciences till 2012, and now with the North China University of Technology). He is in charge of the Cloud Computing Research Center, North China University of Technology. He is also the director of Beijing Key Laboratory on Integration and Analysis of Large-scale Streaming Data. His current research interests include streaming data processing, cloud computing, dependable distributed systems, business process collaboration and management. Dr. Han has undertaken over 40 funded research projects and served as the PI of the China Grid initiative. He has authored or coauthored over 160 papers and 5 books. His team has acquired over 70 intellectual properties, and 15 of them have been transferred to the industry. Dr. Han has organized over 30 academic events as general chairs or program chairs including 12 journal special issues.

Chen Liu received his Ph.D. degree in computer science and technology from the Chinese Academy of Sciences, Beijing, China, in 2007. Since 2012, he has been an Associate Professor at Research Center for Cloud Computing, North China University of Technology, Beijing, China. His research interests include data integration, service modeling, service composition, cloud computing and so on.

Shen Su received his Ph.D. degree in computer science and technology from Harbin Institute of Technology, in 2016. Since 2016, he has been a lecturer at Research Center for Cloud Computing, North China University of Technology, Beijing, China. His research interests include service computing, service composition, inter-domain routing modeling and analysis, sentiment analysis, etc.

Meiling Zhu is currently a Ph.D candidate at the School of Compute Science and Technology of Tianjin University, China. Her research interests include Services Computing, Streaming Data Integration and Analysis.

Zhongmei Zhang is currently a Ph.D candidate at the School of Compute Science and Technology of Tianjin University, China. Her research interests include Services Computing, Streaming Data Integration and Analysis.