Log-based Anomaly Detection of CPS Using a Statistical Method

Yoshiyuki Harada*†, Yoriyuki Yamagata†, Osamu Mizuno*, Eun-Hye Choi†
* Kyoto Institute of Technology, Kyoto, Japan
Email: y-harada@se.is.kit.ac.jp, o-mizuno@kit.ac.jp
†National Institute of Advanced Industrial Science and Technology (AIST), Ikeda, Japan
Email: {yoriyuki.yamagata, e.choi}@aist.go.jp

***Abstract*—Detecting anomalies of a *cyber physical system* (*CPS*), which is a complex system consisting of both physical and software parts, is important because a CPS often operates autonomously in an unpredictable environment. However, because of the ever-changing nature and lack of a precise model for a CPS, detecting anomalies is still a challenging task. To address this problem, we propose applying an *outlier detection* method to a CPS log. By using a log obtained from an actual aquarium management system, we evaluated the effectiveness of our proposed method by analyzing outliers that it detected. By investigating the outliers with the developer of the system, we confirmed that some outliers indicate actual faults in the system. For example, our method detected failures of mutual exclusion in the control system that were unknown to the developer. Our method also detected transient losses of functionalities and unexpected reboots. On the other hand, our method did not detect anomalies that were too many and similar. In addition, our method reported rare but unproblematic concurrent combinations of operations as anomalies. Thus, our approach is effective at finding anomalies, but there is still room for improvement.**

I. Introduction

A *cyber physical system* (*CPS*) is a complex system consisting of both physical and software parts. CPSs have become increasingly ubiquitous in our lives with the advent of machines controlled by software, such as drones, driverless cars, and automatically operated trains. Nowadays, not only are more machines operated automatically, but those machines are also connected by a network and make collective decisions without human intervention.

Our research goal is to find *anomalies* in a CPS by fully automated means. Anomaly detection is especially important for a CPS because anomalies in the autonomous operation of physical objects in the CPS can cause losses of not only property but also even human life [1]. A CPS operates in an ever-changing environment in an ever-changing way. This makes testing a CPS difficult or impossible before the system is deployed. Moreover, a lack of precise models for both a CPS and its environment makes formal analysis burdensome. Therefore, we need a method to detect CPS anomalies in real-time or post mortem and correct the problem.

In this paper, we applied an *outlier* detection technique for CPS anomaly detection. Outlier detection techniques have been extensively studied in the fields of statistics, machine learning, and data mining. An outlier is a data point that is suspected to be generated by a different mechanism than most data points. An outlier is not necessarily an anomaly, but we focused on them because the latter has a good chance to be the former. We applied the *Local Outlier Factor (LOF)* [2], which is widely used for outlier detection of multi-dimensional real-value data, to a log of an automatic aquarium management system and evaluated its effectiveness at anomaly detection. An advantage of the LOF is that no assumption is necessary regarding the distribution of data points.

However, there are several challenges to applying the LOF to CPS anomaly detection. First, a CPS log is a mixture of discrete and real-value data. The LOF is not designed for discrete data. We resolved this problem by mapping log entries to real-value vectors in a high-dimensional space. This mapping is straightforward. Second, the LOF is not designed for a time series. We resolved this problem by concatenating vectorized log entries in a moving window. In this manner, we can detect *position* and *combination* outliers, which are necessary to detect CPS anomalies.

For evaluation, we used an automatic aquarium management system called *Aqua-tan* [3] as a scaled example of a CPS. For experiments, we collected 1,000,000 log entries of Aqua-tan and separated them into 10 chunks. For each chunk, we applied the proposed method. The ELKI tool [4] was used to calculate the *outlier factor* for the LOF algorithm. Finally, we analyzed five log sequences that achieved the highest outlier factors for each chunk by comparing them with a cluster of nearby data points.

Through this analysis, we evaluated the effectiveness of our method at finding CPS anomalies. Our method detected several interesting anomalies. For example, our method detected failures of mutual exclusion in the control system that were unknown to the developer. Our method also detected anomalies such as temporal losses of functionalities, unexpected reboots, and manual interventions to the system. On the other hand, we found several limitations for our method. For example, if there are too many anomalies in sequence, our method detected a sequence of anomalies as inliers. In addition, our method reported rare but unproblematic concurrent combinations of operations as outliers. We concluded that our approach is effective at finding faulty anomalies, but there is still room for improvement.

978-1-5090-6699-5/17 $31.00 © 2017 IEEE
DOI 10.1109/IWESEP.2017.12

II. Related Work

Anomaly detection is a research field that uses diverse methods for diverse applications. For details, please refer to the survey [5] or book [6]. Because the literature on anomaly detection is very extensive, we describe only the work relevant to the CPS, anomaly detection from a software log, and alternative methods for LOF here.

There is extensive literature on anomaly detection of a hybrid system [7]–[11]; all of them presuppose a model of a system. On the other hand, our method does not assume a system model because preparing the model for CPS and its environment is a difficult task.

There is also extensive literature on anomaly detection from a software log [12]–[15]. These papers either assume that a log is purely discrete or real-value data. On the other hand, our method handles a log with both discrete and real-value data.

We handled vectors with very high dimensions in our experiment; thus, we pushed the outlier detection method to its limit. Outlier detection in high-dimensional space is an active research area [16]. In addition to LOF, we attempted to use high contrast subspaces (HiCS) algorithm [17] and the correlation outlier probability (COP) algorithm [18]; however, their computations could not be completed within reasonable time and memory constraints.

III. Preliminary: Local Outlier Factor (LOF)

In this section, we introduce the LOF algorithm based on the work of Aggarwal [6]. The LOF algorithm uses a fixed $k \geq 1$ to determine the *outlier factors* of data points. For each data point x, the outlier factor is determined as follows. Let $D_k(x)$ be the distance of the k-nearest neighbor of x and $L_k(x)$ be the data points within $D_k(x)$ from x. Now, the *reachability distance* $R_k(x, y)$ from x to a data point y is defined as

$$R_k(x, y) = \max\{d(x, y), D_k(y)\} \tag{1}$$

where $d(x, y)$ is the distance between x and y. Typically, the Euclidean distance is used for $d(x, y)$. Then, the *average reachability distance* $AR_k(x)$ is defined as the average of the reachability distance from x to data points in $L_k(x)$:

$$AR_k(x) = \frac{1}{\#L_k(x)} \sum_{y \in L_k(x)} R_k(x, y) \tag{2}$$

where $\#L_k(x)$ is the number of data points in $L_k(x)$. $\#L_k(x)$ may not be equal to k because of a tie. Finally, the outlier factor of x is defined as the average of the ratio between $AR_k(x)$ and $AR_k(y), y \in L_k(x)$:

$$LOF_k(x) = \frac{1}{\#L_k(x)} \sum_{y \in L_k(x)} \frac{AR_k(x)}{AR_k(y)} \tag{3}$$

If the outlier factor for x is large, x is more likely to be an outlier.

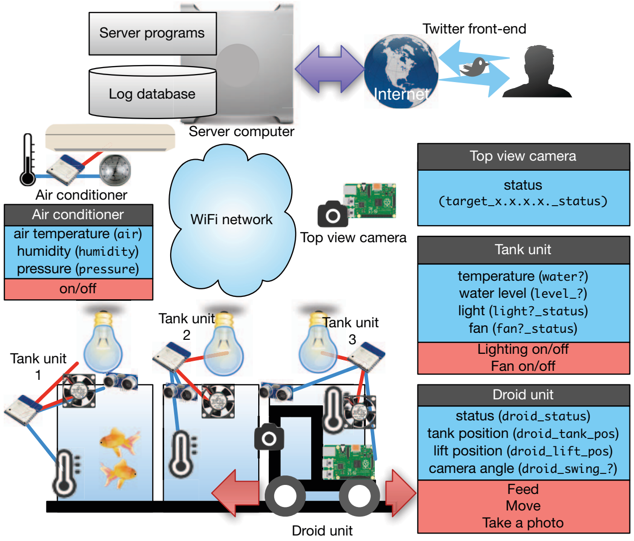

Fig. 1. Aqua-tan: automatic aquarium management system.

IV. Target System and Proposed Method

A. Target System: Aqua-tan

Our target system is the automatic aquarium management system *Aqua-tan*[1] [3]. An overview of the system is shown in Figure 1. Aqua-tan is a CPS for aquarium management with sensors and actuators connected to a WiFi network. Aqua-tan mainly consists of three units: tanks, a droid, and air conditioner control. The tank units monitor thermometers for the water temperature and a supersonic sensor for the water level. They also control the lighting and fans to manage the environment of the aquarium. The droid unit moves between tanks, takes photos of tanks from various positions and angles, and automatically feeds all tanks. The air conditioner control unit switches the air conditioner on or off according to the air temperature of the room. Several other units provide additional functionalities, such as the top view camera server. These units can be controlled by the Twitter front-end, which accepts orders for the tank and droid units in natural language. The Twitter front-end also provides periodic information regarding the tank status and photos.

Aqua-tan generates a log entry for each system command; an example is shown in Fig. 2. Each log entry consists of five columns: the ID, a time stamp, a command name, numerical data, and character string data. The ID is a count of log entries from the beginning. The time stamp is the time when the log entry is recorded. The command name is generated by the system. Aqua-tan has 130 commands that are subdivided into the "Actuator drive", "Sensor value", "Network status", and "Others" (Table I). Each command has either a numerical argument or string data (name). For example, the command "`humidity`", which shows the humidity of the room, has a numerical argument that indicates the humidity measurement

[1] https://se.is.kit.ac.jp/aquarium/

TABLE I
AQUA-TAN COMMANDS.

Class	Kinds	Example
Actuator drive	46	droid_tank_pos, droid_lift_pos, ...
Sensor value	31	cputemp, humidity, water1, ...
Network status	7	target_X.X.X.X_status, ...
Others (Exclude)	46	fridge1_status, location_X, ...

```
39993,"2014-06-06 22:00","air",28,NULL
39994,"2014-06-06 22:00","humidity",30,NULL
39995,"2014-06-06 22:06:18","fan1_status",NULL,"on"
39996,"2014-06-06 22:10","cputemp",49.8,NULL
39997,"2014-06-06 22:10","pressure",994.1,NULL
39998,"2014-06-06 22:10","water2",27.1,NULL
39999,"2014-06-06 22:10","water1",26,NULL
40000,"2014-06-06 22:10","water3",27.8,NULL
40001,"2014-06-06 22:10","air",28,NULL
40002,"2014-06-06 22:10","humidity",30,NULL
```

Fig. 2. Example Aqua-tan log.

of 30, as shown in Fig. 2. On the other hand, the command “`fan1_status`” does not have a numerical argument but the string argument “on”, which indicates that fan 1 is on.

All log entries are stored in a MySQL database. We use the CSV format for the log entries extracted from this database.

B. Proposed Method

Our method for detecting anomalies of the target system consists of six steps.

1) **Pre-processing**: Collect log entries related to the automatic management of the aquarium.
2) **Vectorization**: Convert each log entry to a real-value vector.
3) **Normalization**: Normalize each vector component so that each component has 0 as the average and 1 as the variance.
4) **Windowing**: Concatenate the vectors in the moving window to create a single vector.
5) **LOF**: Calculate the outlier factor for each window using the LOF algorithm.
6) **Post-processing**: Take entries of high outlier factors as anomalies.

Pre-processing. First, we collect log entries that are related to the targeted system. Aqua-tan logs contain entries that are related to the status of mobile terminals and results of sentiment analysis on Twitter statements. These entries are removed by pre-processing because they cause spurious outliers.

Vectorization. Next, each log entry is converted to a real-value vector. The first component of the vector is the time (in milliseconds) that has passed from the previous entry of the same command. The rest of the vector components are determined as follows. For each command in the log entries, we prepare a vector v of the five components that are filled by the real value 0. If a command has the numerical argument a, we put a into the first component $v[0] = a$. If a command has the string argument s, we first take the hash value $h(s)$ of its string argument and compute the remainder $h(s)\%4$ of the hash value when divided by 4. Then, we put $v[1 + h(s)\%4] = 1$. Finally, all vectors for all commands are concatenated in the predetermined fixed order. Note that only one component is non-zero in the vector that is obtained by this procedure, excluding the first component.

Normalization. Third, we normalize vectors by linearly transforming each vector component, to have 0 as the average and 1 as the variance.

Windowing. Fourth, each vector that is obtained with the above method is concatenated by using a moving window of a fixed size. This creates vectors that reflect combinations of commands and their arguments of successive time windows.

LOF. Fifth, the vectors are fed to the LOF algorithm, and outlier factors are computed.

Post-processing. Finally, we consider log entries that belong to windows with high outlier factors as anomalies.

V. EXPERIMENTS AND ANALYSIS

A. Experimental Setup

In our experiment using Aqua-tan log entries, we first created 10 chunks consisting of 100,000 consecutive log entries from the entire log database. We created chunks because the log spanned 2.2 years, and the system changed significantly during this period. Therefore, there was not much sense in processing the entire log at once. Another reason was that the LOF algorithm has a computation time of $O(n \log n) \sim O(n^2)$ depending on the dimension n of the data. Thus, chunking reduces the computation time.

Each chunk was processed independently. Pre-processing, vectorization, normalization, and the taking of a moving window were all done with Python scripts. We vectorized each log entry into 191–301-dimension vector space. The dimension of the vectorized log entry was determined from the number of commands. For example, chunk1 had 196 dimensions, while chunk4 had 201 dimensions. In the windowing phase, we set the window size to 11. The vector concatenated with a window width of 11 had about 2101–3311 dimensional vectors for each window. Then, outlier factors were computed with the standalone tool ELKI [4]. We set $k = 20$ based on the recommendation by Breunig et al. [2]. Fig. 3 visualizes the ELKI output. The circle marker indicates an example from Table 4(a), and the square marker indicates an example from Table 4(b). Finally, the output of ELKI was post-processed with Python scripts. During the post-processing, five windows of the highest outlier factors were extracted, and $k = 20$ log sequences (i.e., k-neighborhoods) that were near the outlier were also recorded for later analysis.

For computing, we used a machine with one Intel(R) Xeon(R) CPU E5-1620 v3, 3.50GHz processor having four cores and 64 GB RAM. It took about 30 hours to complete all tasks of the proposed method.

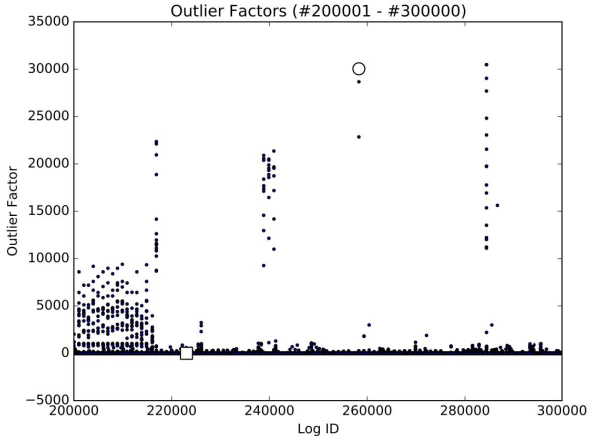

Fig. 3. Time series of outlier factors: from #200000 to #300000.

TABLE II
OUTLIER CLASSIFICATION.

	Cause of outlier	Count
1	Failure of mutual exclusion	2
2	Unexpected reboot	4
3	Failure of single functionality	6
4	Mixture of anomalous and normal events	11
5	Rare mixture of correct behavior	21
6	Configuration change	4
7	Manual operation	2

B. Analysis

For analysis, we classified the extracted 50 log sequences of outliers by the cause of every outlier. Table II summarizes the classification results: seven causes of outliers and the number of outliers in each category. In the following, we explain our analysis results for each category. Example outliers for categories 1–5 are shown in Fig. 4; because of space limitations, we omit examples for categories 6 and 7 in the figure.

1) Failure of Mutual Exclusion: This outlier represents failures of mutual exclusion in the system. The control for moving and feeding by the droid unit requires mutual exclusion. However, the outlier analysis found a violation of mutual exclusion.

Figs. 4(a) and (b) show examples of outliers and inliers, respectively, for this class. In the outlier indicated by Fig. 4(a), the sequence after #258298 was faulty. The control of the moving by the droid unit must be exclusively given by a single process. The control program for the droid indicated critical sections using the "`droid_status,NULL,Operating`" and "`droid_status,NULL,Waiting`" command log entries, shown by the bold face. Any process except one must not use the droid unit after "`droid_status,NULL,Operating`" until "`droid_status,NULL,Waiting`" appears. The inlier log shows this operation between #223032 and #223034 in Fig. 4(b). In the outlier case of Fig. 4(a), although the "`droid_status,NULL,Waiting`" must follow after the log of "`droid_movediff`", in the real log, the log of droid position ("`droid_lift_pos,10,NULL`") followed, and no "`droid_status,NULL,Waiting`" command log appeared after that. The manual feeding operation for tank 2 shown in #258300 caused this conflict. We found that the manual feeding command during the automatic feeding process violated the mutual exclusion for control of the droid unit.

2) Unexpected Reboot: Our target system sometimes rebooted unexpectedly. In the outlier for this category shown in Fig. 4(c), the log entries of #464114 and #464115 are important. There were logs including `light3_ontime` and `light1_ontime` in the early afternoon. These commands adjust the time to turn on the lighting the next morning. Normally, these commands are executed at sunset, as shown in Fig. 4(d). However, upon rebooting, these commands are also executed to adjust the lighting time. In this case, the anomalous executions of `light3_ontime` and `light1_ontime` commands imply that the system was rebooted and initialized.

3) Failure of Single Functionality: This anomaly represents failures of a single function. For example, missing scheduled feeding operations and lost connections of the top view camera are included in this category. This kind of anomaly is usually found as a single log entry.

The aquarium system equips a camera that provides the top view of the aquarium. A Raspberry Pi operates the camera server for this purpose and connects it to the aquarium system by a wireless local area network (LAN). The wireless LAN adapter of this server sometimes lost connection, and the server restarted the LAN adapter on its own. When the outlier (Fig. 4(e)) and its *k*-neighborhood (Fig. 4(f)) are compared, the command log entry `target_X.X.X.X_status,NULL,Lost`, shown in Fig. 4(e), indicated by the bold face, caused the difference, where `X.X.X.X` is an IP address of the camera server. This means that the network connection of the camera server was lost and could not recover the connection by itself.

4) Mixture of Anomalous and Normal Events: Our analysis found interesting sequences in the log. As shown in Fig. 4(h), many sensors successively output incorrect values to the log because both the humidity and water level were 0. Around such sequences, the correct values can be found as outliers. Fig. 4(g) shows a correct sensor reading in an incorrect sensor reading sequence. In this case, the cause of the outlier was the non-anomalous sequence of sensor commands (#464045–#464049), and the anomalous sequence of sensor commands in Fig. 4(h) is recognized as an inlier sequence.

5) Rare Mixture of Correct Behavior: Our method identified rare events during regular operations as an outlier. Fig. 4(i) shows an outlier in this category, and Fig. 4(j) is an example of entries in the nearest cluster from the outlier. They show sequences of regular automatic feeding around noon. If we compare these sequences, we find that the `fan3_status` command at #15196 in Fig. 4(i) caused the difference. This

```
258290,"2015-01-01 12:03:22","droid_movediff",NULL,"2,-3"
258291,"2015-01-01 12:03:44","droid_status",NULL,"Waiting"
258292,"2015-01-01 12:03:44","droid_tank_pos",12,NULL
258293,"2015-01-01 12:03:44","droid_lift_pos",6,NULL
258294,"2015-01-01 12:03:57","feed_tank_2",4800,NULL
258295,"2015-01-01 12:04:08","droid_swing_h",150,NULL
258296,"2015-01-01 12:04:08","droid_swing_v",150,NULL
258297,"2015-01-01 12:04:08","droid_status",NULL,"Operating"
258298,"2015-01-01 12:04:11","droid_movediff",NULL,"4,4"
258299,"2015-01-01 12:06:12","droid_lift_pos",10,NULL
258300,"2015-01-01 12:08:07","feed_tank_2",1000,NULL
```

(a) Outlier 1: Failure of mutual exclusion.

```
223025,"2014-11-28 12:02:30","droid_movediff",NULL,"2,-3"
223026,"2014-11-28 12:02:59","droid_status",NULL,"Waiting"
223027,"2014-11-28 12:02:59","droid_tank_pos",12,NULL
223028,"2014-11-28 12:02:59","droid_lift_pos",6,NULL
223029,"2014-11-28 12:03:22","feed_tank_2",4800,NULL
223030,"2014-11-28 12:03:40","droid_swing_h",150,NULL
223031,"2014-11-28 12:03:41","droid_swing_v",150,NULL
223032,"2014-11-28 12:03:42","droid_status",NULL,"Operating"
223033,"2014-11-28 12:03:45","droid_movediff",NULL,"4,4"
223034,"2014-11-28 12:05:03","droid_status",NULL,"Waiting"
223035,"2014-11-28 12:05:04","droid_tank_pos",16,NULL
```

(b) *k*-neighborhood of outlier 1.

```
464104,"2015-06-28 13:20","level_3",0,NULL
464105,"2015-06-28 13:20","lightning",NULL,NULL
464106,"2015-06-28 13:20","humidity",0,NULL
464108,"2015-06-28 13:30","level_3",0,NULL
464109,"2015-06-28 13:30","lightning",NULL,NULL
464110,"2015-06-28 13:30","humidity",0,NULL
464111,"2015-06-28 13:37:41","light3_status",NULL,"on"
464112,"2015-06-28 13:37:41","light2_status",NULL,"on"
464113,"2015-06-28 13:37:41","light1_status",NULL,"on"
464114,"2015-06-28 13:37:41","light3_ontime",NULL,"06:35"
464115,"2015-06-28 13:37:41","light1_ontime",NULL,"06:45"
```

(c) Outlier 2: Unexpected reboot.

```
408976,"2015-05-22 17:40","air",26.2,NULL
408977,"2015-05-22 17:40","humidity",22.4,NULL
408978,"2015-05-22 17:50","pressure",1004,NULL
408979,"2015-05-22 17:50","water2",26.6,NULL
408980,"2015-05-22 17:50","level_3",9,NULL
408981,"2015-05-22 17:50","water1",24.9,NULL
408982,"2015-05-22 17:50","water3",25.8,NULL
408983,"2015-05-22 17:50","air",26.2,NULL
408984,"2015-05-22 17:50","humidity",22.4,NULL
408985,"2015-05-22 17:58:02","light1_status",NULL,"off"
408986,"2015-05-22 17:58:02","light1_ontime",NULL,"06:49"
```

(d) *k*-neighborhood of outlier 2.

```
630999,"2015-11-01 12:05:13","droid_status",NULL,"Waiting"
631000,"2015-11-01 12:05:13","droid_tank_pos",12,NULL
631001,"2015-11-01 12:05:14","droid_lift_pos",5,NULL
631002,"2015-11-01 12:05:40","feed_tank_2",10000,NULL
631003,"2015-11-01
     12:05:40","target_X.X.X.X_status",NULL,"Lost"
631004,"2015-11-01 12:05:47","droid_swing_h",0,NULL
631005,"2015-11-01 12:05:48","droid_swing_v",-20,NULL
631006,"2015-11-01 12:05:48","droid_status",NULL,"Operating"
631007,"2015-11-01 12:05:51","droid_movediff",NULL,"4,2"
631008,"2015-11-01 12:06:33","droid_status",NULL,"Waiting"
631009,"2015-11-01 12:06:33","droid_tank_pos",16,NULL
```

(e) Outlier 3: Failure of single functionality.

```
623096,"2015-10-26 12:05:27","droid_status",NULL,"Waiting"
623097,"2015-10-26 12:05:27","droid_tank_pos",12,NULL
623098,"2015-10-26 12:05:27","droid_lift_pos",5,NULL
623099,"2015-10-26 12:05:58","feed_tank_2",10000,NULL
623100,"2015-10-26 12:06:15","droid_swing_h",0,NULL
623101,"2015-10-26 12:06:15","droid_swing_v",-20,NULL
623102,"2015-10-26 12:06:15","droid_status",NULL,"Operating"
623103,"2015-10-26 12:06:19","droid_movediff",NULL,"4,2"
623104,"2015-10-26 12:07:08","droid_status",NULL,"Waiting"
623105,"2015-10-26 12:07:08","droid_tank_pos",16,NULL
623106,"2015-10-26 12:07:08","droid_lift_pos",7,NULL
```

(f) *k*-neighborhood of outlier 3.

```
464045,"2015-06-28 11:00","water3",27.1,NULL
464046,"2015-06-28 11:00","water1",25.9,NULL
464047,"2015-06-28 11:00","lightning",-1,NULL
464048,"2015-06-28 11:00","air",27.5,NULL
464049,"2015-06-28 11:00","humidity",41.3,NULL
464051,"2015-06-28 11:10","level_3",0,NULL
464052,"2015-06-28 11:10","lightning",NULL,NULL
464053,"2015-06-28 11:10","humidity",0,NULL
464055,"2015-06-28 11:20","level_3",0,NULL
464056,"2015-06-28 11:20","lightning",NULL,NULL
464057,"2015-06-28 11:20","humidity",0,NULL
```

(g) Outlier 4: Mixture of anomalous and normal events.

```
464052,"2015-06-28 11:10","lightning",NULL,NULL
464053,"2015-06-28 11:10","humidity",0,NULL
464055,"2015-06-28 11:20","level_3",0,NULL
464056,"2015-06-28 11:20","lightning",NULL,NULL
464057,"2015-06-28 11:20","humidity",0,NULL
464059,"2015-06-28 11:30","level_3",0,NULL
464060,"2015-06-28 11:30","lightning",NULL,NULL
464061,"2015-06-28 11:30","humidity",0,NULL
464063,"2015-06-28 11:40","level_3",0,NULL
464064,"2015-06-28 11:40","lightning",NULL,NULL
464065,"2015-06-28 11:40","humidity",0,NULL
```

(h) *k*-neighborhood of outlier 4.

```
15191,"2014-05-17 12:02:54","feed_tank_1",300,NULL
15192,"2014-05-17 12:03:05","droid_status",NULL,"Operating"
15193,"2014-05-17 12:03:38","droid_status",NULL,"Waiting"
15194,"2014-05-17 12:03:38","droid_tank_pos",12,NULL
15195,"2014-05-17 12:03:38","droid_lift_pos",5,NULL
15196,"2014-05-17 12:03:39","fan3_status",NULL,"off"
15197,"2014-05-17 12:03:47","feed_tank_2",4000,NULL
15198,"2014-05-17 12:04:00","droid_status",NULL,"Operating"
15199,"2014-05-17 12:04:37","droid_status",NULL,"Waiting"
15200,"2014-05-17 12:04:37","droid_tank_pos",16,NULL
15201,"2014-05-17 12:04:37","droid_lift_pos",7,NULL
```

(i) Outlier 5: Rare mixture of correct behavior.

```
4267,"2014-05-08 12:03:35","droid_status",NULL,"Operating"
4268,"2014-05-08 12:04:11","droid_status",NULL,"Waiting"
4269,"2014-05-08 12:04:11","droid_tank_pos",12,NULL
4270,"2014-05-08 12:04:11","droid_lift_pos",5,NULL
4271,"2014-05-08 12:04:14","droid_swing_h",120,NULL
4272,"2014-05-08 12:04:14","droid_swing_v",125,NULL
4273,"2014-05-08 12:04:20","feed_tank_2",4000,NULL
4274,"2014-05-08 12:04:30","droid_status",NULL,"Operating"
4275,"2014-05-08 12:05:06","droid_status",NULL,"Waiting"
4276,"2014-05-08 12:05:06","droid_tank_pos",16,NULL
4277,"2014-05-08 12:05:06","droid_lift_pos",7,NULL
```

(j) *k*-neighborhood of outlier 5.

Fig. 4. Examples of outliers and their *k*-neighborhoods.

means that external fan for the tank 3 starts cooling. Because it is very rare that such unrelated commands are observed during feeding, this was detected as an outlier.

6) Configuration Change: Changes to system settings are detected as an outlier. Such changes are usually done to calibrate the feeding and camera positions, adjust the air conditioner target temperature, and so on.

7) Manual Operation: Because most operations in the system are automatic, manual operation can be identified as an outlier. For example, early feeding for a specific tank is detected as an outlier. In this aquarium system, manual feeding rarely happens because the system allows a few feeds per day for each tank. This is why our method detects unusual feeding commands as an outlier.

C. Observation

We expected that our method would be able to detect a single rare entry of a log, such as the failure of a single functionality. Surprisingly, our method can also detect a complex anomaly that involves several commands, such as the failure of mutual exclusion. This is the virtue of using a moving window rather than a single log entry to perform outlier detection. The identification of a real unknown bug shows the capability of our method.

On the other hand, some kinds of anomalies cannot be detected as outliers by our method. This is because, if many anomalies exist in the log and are similar (e.g., all sensor readings are 0), they form highly agglomerated clusters and thus become inliers according to our method. Interestingly, outliers detected by our method often showed partly normal and partly anomalous behavior. This is because they came near a cluster of anomalous values and thus were partly anomalous but differed from them by being partly normal. Such sequences can be an entry point to anomalous behavior. Thus, finding such sequences can be useful.

Another interesting observation is that, for each chunk, the obtained outliers had quite different characteristics. For example, outliers of chunk 3 consisted of sensor errors, while outliers of chunk 8 were related to sensing and managing the water level. This is useful for finding problems that the target system faces in each development phase.

VI. Conclusions and Future Work

We applied an outlier detection method, LOF, to find anomalies in a CPS log and evaluated the usefulness of our method by analyzing outliers it detected in an aquarium management system. Based on our analysis results, the proposed method can detect many interesting events, such as failures of mutual exclusion, unexpected reboots, and failures of single functionality. It also can detect manual operation of the system, which is usually harmless but can indicate a malicious attempt to control the system.

On the other hand, our method is not suitable for detecting anomalies that occur many times in similar forms; such anomalies form a cluster and are judged as inliers by our method. Still, even in such a case, our method can detect log sequences that are partially similar to such anomalies but partially correct. Such log sequences can be entry points to anomalous modes in the system and thus can be useful for finding the causes of anomalies.

Future work will involve comparing the LOF with other outlier detection methods for detecting CPS anomalies. Our method currently cannot detect anomalies that frequently occur and form agglomerated clusters. This could be overcome by using other outlier detection methods, such as a robust principal component analysis (PCA)-based method [19] and isolation forests [20].

Acknowledgment

We thank Hisashi Kashima for his helpful comments. We would like to thank Editage for English-language editing.

References

[1] D. Yadron and D. Tynan, “Tesla driver dies in first fatal crash while using autopilot mode,” *The Guardian*, July 1, 2016.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “LOF: Identifying Density-Based Local Outliers,” *Proc. of SIGMOD*, pp. 1–12, 2000.

[3] O. Mizuno, “Auto-feeding and moving aquarium management droid — everybody’s Raspberry Pi contest, the grand prix work,” *Nikkei Linux (in Japanese)*, vol. 17, no. 4, pp. 77–83, 4 2015.

[4] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek, “A framework for clustering uncertain data,” *PVLDB*, vol. 8, no. 12, pp. 1976–1979, 2015.

[5] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly Detection : A Survey,” *ACM computing surveys*, 2009.

[6] C. Aggarwal, *Outlier Analysis*. Springer Publishing Company, 2015.

[7] V. Verma, G. Gordon, R. Simmons, and S. Thrun, “Real-time fault diagnosis,” *IEEE Robotics and Automation Magazine*, vol. 11, no. 2, pp. 56–66, 2004.

[8] S. Narasimhan and G. Biswas, “Model-Based Diagnosis of Hybrid Systems,” *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 37, no. 3, pp. 348–361, 2007.

[9] M. W. Hofbaur and B. C. Williams, “Mode estimation of probabilistic hybrid systems,” *Lecture Notes in Computer Science*, vol. 2289, pp. 253–266, 2002.

[10] F. Zhao, X. Koutsoukos, H. Haussecker, J. Reich, and P. Cheung, “Monitoring and Fault Diagnosis of Hybrid Systems,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 35, no. 6, pp. 1225–1240, 2005.

[11] M. W. Hofbaur and B. C. Williams, “Hybrid estimation of complex systems.” *IEEE transactions on systems, man, and cybernetics, Part B: Cybernetics*, vol. 34, no. 5, pp. 2178–2191, 2004.

[12] B. Russo, G. Succi, and W. Pedrycz, “Mining system logs to learn error predictors: a case study of a telemetry system,” *Empirical Software Engineering*, vol. 20, no. 4, pp. 879–927, 2014.

[13] S. He, J. Zhu, P. He, and M. R. Lyu, “Experience Report : System Log Analysis for Anomaly Detection,” *Proc. of ISSRE*, 2016.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, M. I. Jordan, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, “Detecting Large-Scale System Problems by Mining Console Logs,” *Proc. of SOSP*, pp. 117–131, 2009.

[15] Y. Chen, C. M. Poskitt, and J. Sun, “Towards Learning and Verifying Invariants of Cyber-Physical Systems by Code Mutation,” *Proc. of FM*, pp. 155–163, 2016.

[16] C. Aggarwal, “High-Dimensional Outlier Detection: The Subspace Method,” *Outlier Analysis*, 2013.

[17] F. Keller, E. Müller, and K. Böhm, “HiCS: High contrast subspaces for density-based outlier ranking,” *Proc. of ICDE*, no. 1, pp. 1037–1048, 2012.

[18] H. P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, “Outlier detection in arbitrarily oriented subspaces,” *Proc. of ICDM*, vol. 1, no. Icdm, pp. 379–388, 2012.

[19] E. J. Candès, X. Li, Y. Ma, and J. Wright, “Robust principal component analysis?” *JACM*, vol. 58, no. 3, p. 11, 2011.

[20] F. T. Liu, K. M. Ting, and Z. Zhou, “Isolation forest,” in *Proc. of ICDM*. IEEE, 2008, pp. 413–422.