WIP: Sysnif: Constructing Workflow from Interleaved Logs in Intelligent IoT System

Zongming Jin[1], Xueshuo Xie[1], Yaozheng Fang[1], Zhaolong Jian[1], Ye Lu[1,3,*], Guangying Li[2,*]
[1] College of Cyber Science, Nankai University, Tianjin, China
[2] Cyberspace Administration of Tianjin, China
[3] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{zongming_jin, xueshuoxie, fyz, jianzhaolong}@mail.nankai.edu.cn, luye@nankai.edu.cn, ligying@163.com

***Abstract*—The massive smart devices in intelligent IoT can be broken due to malicious attacks and system failures. As a non-intrusive method, workflows mined from system logs facilitate administrators to quickly locate and diagnose anomalies in time. System logs are usually interleaved since there are lots of concurrent and asynchronous operations and executions on large scale IoT devices. Consequently, it is so challenging to construct an adaptive workflow from these logs and realize the real-time anomaly detection. To meet this challenge, in this paper, we propose a two-stage workflow construction approach named Sysnif, which includes offline construction and online adjustment. First, the window-based dependence computing method is employed to obtain the context of execution paths. Second, a weight-greedy algorithm is designed to denoise the interleaved system logs effectively. Third, in order to match system mechanism variation, the online micro-iteration adjusting algorithm is presented to update the workflow model. Experiment results highlight that Sysnif can outperform state-of-the-art methods, such as Logsed, on dataset of OpenStack logs by 22.4% on recall, meanwhile maintaining the same precision roughly. Sysnif can achieve an average precision and recall of 93.8% and 94.7%, respectively.**

***Index Terms*—Workflow, Interleaved Logs, Dependence computing, Micro-iteration adjusting**

I. Introduction

Modern intelligent IoT systems become extremely complex, including thousands of components and providing services for multiple users concurrently. For administrators, monitoring from system logs is an important way to catch operating status and maintain system running normally. With the increase of IoT system scale and complexity, a large number of logs generated has achieved the terabyte level per day [1]. Manually processing these logs is too hard to understand what happens about systems over time. Workflow mined from system logs is a non-intrusive method to facilitate system maintenance, which can help administrators understand system behaviors and monitor the validity of system operations. However, it is challenging that workflow is usually not available due to system imperfect documents and specifications [2]. It is also difficult to build an adaptive workflow model from the interleaved logs, since the running mechanism of IoT system is variable and polymorphous.

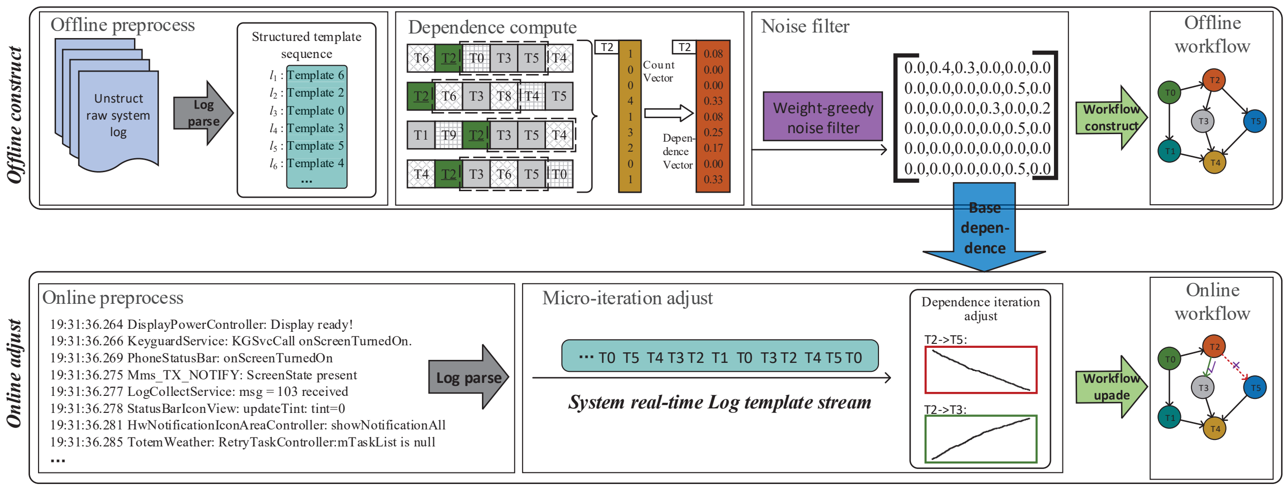
It is a common practice to record system run-time information in logs in detail which facilitate administrators or researchers to understand system behaviors and then locate possible problems [3], [4]. A typical raw log entry is shown as following example, containing *timestamp*, *verbosity level* and *raw message content*.

```
2020-02-10 20:38:31,5186 INFO dfs.DataNode$DataXceiver:
Receiving block blk_6853481264720481267 src:
/10.210.11.53:48251 dest: /10.210.11.53:50754
```

The raw message content can be divided into two parts: *constant* part and *variable* part. The constant part is the fixed plain text which displays every event type in log messages. The variable part represents the system run-time information changing over time. For log entries, the *template* is responsible to the constant part and the *parameter* represents the variable part. In the example, the template can be denoted as "$Receiving\ block * src : * dest : *$", and the parameter is marked using $*$. Since the template represents specific state when system running, it is used as the node in the workflow model.

However, concurrent and asynchronous of the system operations result in log interleaving, which is often disorderly by intuition. In general, log interleaving can introduce context missing and noise problems, which make it hard to separate the logs of different tasks. But in principle, to construct workflow models, administrators or researchers have to separate logs first for extracting program execution paths. Some previous works [2], [5] assume that each log entry has an identifier that uniquely identifies the specific execution. Nevertheless, system logs are different from application logs, which cannot obtain unique identification information. Besides, the variability and polymorphism of system bring the model aging problem. However, some works based on data mining or machine learning is carried out offline, such as [6] and [7], without considering the adverse effects of model aging. What's more, there is commercially security information and event management software, such as Splunk [8], that is used in processing log data, but they require instrumentation in specific system components to generate traces for further analysis.

To address the above problems, we construct workflow models from system logs for widely applicable and non-intrusive monitor to the IoT system. In this paper, we propose a two-stage approach named Sysnif to construct the workflow



```
1      0.08
0      0.00
0      0.00
4      0.33
1      0.08
3      0.25
2      0.17
0      0.00
1      0.33
```

```
0.0,0.4,0.3,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.5,0.0
0.0,0.0,0.0,0.3,0.0,0.2
0.0,0.0,0.0,0.0,0.5,0.0
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.5,0.0
```

```
19:31:36.264 DisplayPowerController: Display ready!
19:31:36.266 KeyguardService: KGSvcCall onScreenTurnedOn.
19:31:36.269 PhoneStatusBar: onScreenTurnedOn
19:31:36.275 Mms_TX_NOTIFY: ScreenState present
19:31:36.277 LogCollectService: msg = 103 received
19:31:36.278 StatusBarIconView: updateTint: tint=0
19:31:36.281 HwNotificationIconAreaController: showNotificationAll
19:31:36.285 TotemWeather: RetryTaskController:mTaskList is null
...
```

```
··· T0 T5 T4 T3 T2 T1 T0 T3 T2 T4 T5 T0
```

Fig. 1. Sysnif main procedures: offline workflow construction and online adjust update.

offline and adjust the workflow online. At the offline stage, we build a basic workflow model relied on interleaved system logs. To deal with context missing caused by logs interleaving, we employ the variable window-based dependence computing method and design a weight-greedy algorithm to find the true neighbors and filter the noise. Then, at the online stage, we present a micro-iteration algorithm to match system mechanism variation by updating the offline model. In summary, this paper makes the following contributions:

- Employ the variable window-based method for dependence computing to prevent context missing. The true neighbors of the current entry in interleaved logs can be covered through variable windows.
- Design an effective weight-greedy noise filter algorithm, which can select the true successor among candidates in the window and remove noise branches.
- Present a micro-iteration adjusting algorithm to be adaptable for system variation. This algorithm can not only obtain the real-time workflow models, but also avoid a lot of resource consumption when constructing workflow once again.

II. Workflow construct

In this section, we give a detailed description of Sysnif. As shown in Fig. 1, the main procedure consists of the offline stage and the online stage. To overcome the context missing and noise issues, we employ the variable window-based method to compute template dependence and design a weight-greedy algorithm to filter noise. The micro-iteration algorithm is presented to update the offline workflow model with system real-time running.

A. Window setting guidance

Logs belonging to the same execution will not be continuous in the log file as the context missing. Thus, it is inadequate to only check the direct predecessor or successor when building the workflow model. No matter how context missing, the true predecessor or successor of the current log entry belonging to the same execution will appear nearby. In this paper, Sysnif sets an n-length window to cover the true neighbors.

When setting a window to find the true successor of the current template, the rest entries in the window have no relationship with this current template. The rest entries are the noise although they may be the running track of other executions. A proper window length can ensure the successor discovery ability and also reduce the noise introduced as much as possible. The calculation of the optimal window length is an NP-hard problem, hence it is reasonable to set an optimal close window length. In practice, the proper window length setting can start from the number of executions until the length of the shortest task.

B. Dependence computing

Dependence is used to measure the bond between templates. As shown in Fig. 1's Dependence compute part, Count Vector (C_V) is firstly calculated where the value is the template occurrence count respectively within the window after the current template in the log sequence. Although the system logs are interleaved, the true neighbors of the template entry will appear nearby. The true successor of the reference template will finally appear in a short period while noise template entries randomly appear. Therefore, true successor entries of the current template have a higher frequency of appearing within the window and the frequency can represent the dependence of two templates. The Dependence Vector (D_V) can be computed through formula (1).

$$D\_V_i[j] = C\_V_i[j]/sum(C\_V_i) \tag{1}$$

Where D_V_i and C_V_i represent Ti's Dependence Vector and Count Vector respectively.

There are two types of dependence: true dependence and noise dependence. True dependence is naturally caused by the time-series order of the ground template sequence, but noise dependence is brought by the random interleaving of other execution log entries. Intuitively, the program just needs to set a filtering threshold of Θ. For each template Ti, if $Dependence(i, j) > \Theta$, there is a true dependence between Ti and Tj, on the contrary, it is noise dependence that needs to be filtered. This method cannot handle the multi-branch scenario, which leads to dependence dispersion. Since the frequency derived from the main branch spreads to multiple branches, the multi-branch dependence is likely to be judged as noise.

C. Weight-greedy noise filter

To deal with the noise issue in the dependence vector, the weight-greedy algorithm is designed to filter noise. Firstly, the feature enhancement is used to solve the problem of multi-branch feature dispersion. The feature enhancement is to combine multiple dependence to form a new feature and use the enhanced feature to do jointly noise filtering. Secondly, the smallest subset which just consists of true dependence without noise is computed. Weight-greedy noise filter algorithm tries 1-union, 2-union, 3-union, ... n-union until the jointly enhanced feature can meet the dependence requirement.

Algorithm 1 Weight-greedy noise filter

```
Input: a Dependence Vector $D\_V$.
Output: the smallest subset and the pure dependence vector.
Global: a significance factor $\gamma$.
 1: $S_{smallest} \leftarrow \{\}$;
    // The smallest subset is initialized as the empty set.
 2: $\|D\_V\|_F = \sqrt{\sum_i D\_V_i^2}$;
    // The Frobenius norm of $D\_V$.
 3: $L = len(D\_V)$;
    // The number of templates.
 4: $dependence\_array = list(D\_V)$;
 5: $sort(dependence\_array)$;
 6: for $i = 0; i < L; i++$ do
 7:     $S_{smallest}.append(dependence\_array_i)$;
 8:     if $\|S_{smallest}\|_F / \|D\_V\|_F > \gamma$ then
 9:         break;
10:     end if
11: end for
12: for $i = 0; i = L; i++$ do
13:     if not $D\_V_i$ in $S_{smallest}$ then
14:         $D\_V_i = 0$;
15:     end if
16: end for
17: return $S_{smallest}$ and $D\_V$
```

Algorithm 1 shows weight-greedy noise filter details. At first, the smallest subset is initialized as an empty set, where to place the true successor templates. Then Frobenius norm of D_V is computed to represent the dependence vector information. A significance factor γ is defined globally. To make greedy choices, sorting is performed on D_V. Then the algorithm continually appends the dependence to the smallest subset according to the sorted sequence until the proportion reaches γ. Finally, the values in D_V that are not in the smallest subset are cleared as noise and return the results. A basic workflow model can be built from pure D_V easily where the non-zero value means that there is an edge between two templates.

D. Workflow model update

The system mechanism has different state migration over time, or even completely shifts to a new running model that has never appeared before. The workflow model established offline can be updated by capturing changes in the system logs to match the variation of the system mechanism. There are two situations for workflow update, namely, real-time workflow model fine-tuning and periodic model reconstruction. Workflow fine-tuning uses the real-time log stream to adjust the weights of the edges in the offline workflow model and dynamically adds or removes the state transition edges. Since the system mechanism may completely shift to a new running model, the periodic check and reconstruction are also adopted to avoid multitudinous meaningless fine-tuning.

The offline workflow model is constructed from pure D_V after noise filtering, where the transition weights correspond to the dependence value in D_V. In the online fine-tuning stage, the weight micro-iteration algorithm continuously adjusts the dependence from the log stream. If the dependence value reaches the upper or lower limit, the online update method needs to add or remove the transition edge.

As shown in algorithm 2, the micro-iteration adjust algorithm continually checks the real-time log stream and fine-tune the dependence value iteratively. The online model is initialized as the static workflow model established offline, and obtains all dependence values meanwhile. Then the algorithm continually takes L length $Fragment$, if the dependence pair of the online model can be found in the $Fragment$, add a micro-iteration step μ to the dependence. On the contrary, subtract μ accordingly. What's more, the upper limit $Limit_u$ and the lower limit $Limit_l$ are set to judge whether to add or remove a particular edge between template nodes. When the dependence value exceeds the $Limit_u$ or lowers than the $Limit_l$, the corresponding transition edge needs to be added or removed.

III. EVALUATION

Five groups of experiments data sets are collected from OpenStack to evaluate the workflow construction performance of Sysnif. These experiments test the potential factors that affect the results of accuracy and efficiency: the tasks concurrently executing number. The more tasks running concurrently, the higher the interleaving complexity of logs. The concurrent task number is set from 2 to 6. We choose state-of-the-art Logsed [9] based on clustering as our baseline and the comparison metrics are standard Precision/Recall. The precision is the fraction of mined edges that are in the ground truth workflow model and the recall is the fraction of the ground truth workflow model edges that are mined.

The Precision and Recall compare experiment results are illustrated in Fig. 2. Logsed has high precision but the recall is unsatisfied badly as the number of concurrency increases. The branches in the OpenStack VM life cycle model will

Algorithm 2 Micro-iteration adjust

Input: a log stream $Stream_{log}$ and a basic offline workflow model $Model_{offline}$.
Output: the real-time online workflow model $Model_{online}$.
Global: a fragment length L, a micro-iteration step μ, and the dependence upper limit and lower limit $Limit_u$, $Limit_l$.

```
 1: $Model_{online} \leftarrow Model_{offline}$
    // The $Model_{online}$ is initialized as $Model_{offline}$
 2: while True do
 3:   $D\_V = getD\_V(Model_{online})$
 4:   $logFragment = getleastNEntries(L, Stream_{log})$
 5:   for $(Predec, Succes)\ in\ Model_{online}$ do
 6:     if $(Predec, Succes)\ in\ logFragment$ then
 7:       $D\_V_{Predec,Succes} += \mu$
 8:     else
 9:       $D\_V_{Predec,Succes} -= \mu$
10:     end if
11:     if $D\_V_{Predec,Succes} > Limit_u$ then
12:       $addEdgeBetween(Predec, Succes)$
13:     end if
14:     if $D\_V_{Predec,Succes} < Limit_l$ then
15:       $removeEdgeBetween(Predec, Succes)$
16:     end if
17:   end for
18: end while
```

lead to dependence dispersion and Logsed filters out some true dependence as noise. While Sysnif employs the weight-greedy noise filter covering that shortage and has a better performance. Sysnif can achieve an average precision and recall of 93.8% and 94.7%, respectively.

We also carry out a comparison experiment of direct construction of 3k logs and 2k logs construction + 1k logs micro-iteration and the Precision/Recall results are 0.948/0.935 and 0.953/0.929. In this experiment, the window length L is set to 5 and the micro-iteration step μ is set to 0.008. The upper limit $Limit_l$ and the lower limit $Limit_u$ are set to 0.1 and 0.9. It can be concluded that construction + micro-iteration can achieve the accuracy level of direct construction. However, micro-iteration works online that outputs the real-time workflow model with the system running and avoids the massive resource consumption of workflow re-construction.

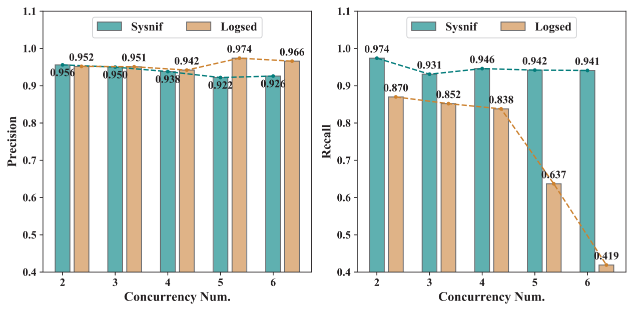


Fig. 2. The Precision/Recall comparison of Sysnif and Logsed.

Sysnif is based on statistical computing with very few iterations. The time complexity of D_V computation and sorting procedure in weight-greedy noise filter are $O(n)$ and $O(Nlog(N))$ respectively, where n is the log length and N is the template number which is much smaller than n. In i5-8300H CPU and 8GB Memory hardware environment, Sysnif's time consumption of processing 10k logs is 3.95s. It can be concluded that Sysnif can achieve $seconds\ per\ 10k$ logs processing level.

IV. Conclusion and Future Work

In this paper, we propose Sysnif to construct the workflow model from interleaved system logs which can facilitate system maintenance in intelligent IoT. To deal with the context missing and noise problems caused by interleaved logs, and the model aging issues lead by the system running mechanism variation, Sysnif utilizes variable window-based dependence computing, weight-greedy noise filter, and micro-iteration adjust update. Finally, the workflow model can be constructed from offline interleave logs and adjusted to match system mechanism variation. The workflow model can be applied in many fields, such as intelligent IoT system maintenance, distribute system tracing, and cloud computing anomaly detection, which will be studied in the future.

Acknowledgment

This work is partially supported by the National Key Research and Development Program of China (2018YFB2100300), the CERNET Innovation Project (NGII20180306, NGII20190402), and sponsored by Zhejiang Lab (2021KF0AB04).

References

[1] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 102–111.

[2] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining program workflow from interleaved traces," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 613–622.

[3] X. Xie, Z. Jin, J. Wang, L. Yang, Y. Lu, and T. Li, "Confidence guided anomaly detection model for anti-concept drift in dynamic logs," *Journal of Network and Computer Applications*, vol. 162, p. 102659, 2020.

[4] X. Xie, Z. Jin, Q. Han, S. Huang, and T. Li, "A confidence-guided anomaly detection approach jointly using multiple machine learning algorithms," in *International Symposium on Cyberspace Safety and Security*. Springer, 2019, pp. 93–100.

[5] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 ninth IEEE international conference on data mining*. IEEE, 2009, pp. 149–158.

[6] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 215–224.

[7] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.

[8] https://www.splunk.com/.

[9] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu, "Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 447–455.