Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 352–357

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

# Trace reconstruction in system logs for processing with process mining

Jasper Paul Jürgensen[a,*]

[a]FH Oberösterreich, Softwarepark 11, 4232 Hagenberg, Austria

**Abstract**

System logs contain information about the behavior of the system. To perform anomaly detection based on this log data, the normal behavior of the system must be learned. With process discovery a process model can be learned from log data. However, system logs do not fulfill the requirements that process discovery algorithms place on log data. To solve this problem, trace reconstruction is used to extract traces from the log data, which are then used for process discovery in a further step. This research proposal therefore proposes combining different methods for grouping log messages. A short experiment shows that the reconstruction of traces from simple log data is possible in principle.



*Keywords:*
process mining; trace reconstruction; anomaly detection

## 1. Introduction

Today's systems produce a whole range of log data. This log data contains information about the behavior of this system and which tasks and services were performed. The sheer volume of this log data can often no longer be analyzed by humans alone. For this reason, automated methods are increasingly used to analyze log data. First of all, there is misuse detection, which is rule-based. The rules used by misuse detection are often created manually.
Due to the increasing complexity of the log data, the effort required to create and maintain these rule sets increases enormously. Anomaly detection is used to counteract this problem. Anomalies are patterns in data that do not correspond to a previously described normal behavior. To apply anomaly detection, normal behavior must first be defined or learned. Because there are different types of anomalies, there are also different methods for detecting each type of anomaly. Execution path anomalies are one type of anomaly.
To find execution path anomalies is especially helpful if the system has a high degree of automation. These systems

* Corresponding author. Tel.: +43-677-62189818.
*E-mail address:* jasper.juergensen@students.fh-hagenberg.at

1877-0509 
10.1016/j.procs.2021.01.173

usually do not have a lot of interaction with human users, so the execution path is repetitive and does not change much. Systems with these characteristics are often found in production environments. The software running on such systems in production is often outdated and thus more vulnerable to classic attacks such as buffer overflows than modern systems. Detecting execution path anomalies can therefore help to detect these types of attacks on these systems.
One way to learn possible execution paths from log data is process mining, especially process discovery. In order to be able to learn the process model from the log data using process discovery, some requirements are placed on the log data. One log data requirement for process discovery is the existence of transaction logs [10]. The individual entries in such a log are events. These events must have at least one activity or event type (definable step in a process), a case (process instance), and a timestamp [9]. System logs do not usually have all of these properties. A timestamp is present in most system logs, but explicit information about the event type or case is usually not available.

The overall goal is to extract patterns from log data, which enable the detection of execution path anomalies. For this purpose, recurring patterns from events are to be found in the log files. The detected patterns should be available in a readable and comprehensible form for a human analyst.
As described earlier, the log data for the use of process mining to identify the patterns are not available in a format required by process mining algorithms. The goal of this approach is to close this gap between system logs and a process mining approach. For this purpose, an event type must first be determined for each event in the log. Additionally, events have to be grouped in a way that a group of events represents a case.

## 2. Related work

The process to extract patterns from log files consists of three steps: Log abstraction, log grouping, and simplification.
Two commonly used log abstraction methods are described by Fu et al. [4] and Jiang et al. [5]. The approach by Fu et al.[4] consists of four steps: Erasing parameters by empirical rules, raw log key clustering, group splitting and log key extraction. The approach by Jiang et al. [5] is based on clone detection. Another possible approach to log abstraction is to search the source code for log templates [11].
The log grouping step is the step where log lines which belong to one case are grouped into one group. Since many current works use Hadoop logs, many methods allow log grouping directly based on the task-id in the log data [8, 6, 7, 11]. A different approach is proposed by Breier and Branišová [2]. Their approach groups log messages based on the correlation of attributes. Log messages within the same session and if the IP addresses and ports are identical are grouped into one group. The approach proposed by Lou et al. [7] also uses the correlation between attribute values to group log messages. A different approach is proposed by Du et al. [3]. They use a LSTM-based neural network. The network predicts the next log messages based on the log messages already seen. Using this predictions it is possible to generate log sequences. A second approach proposed by Du et al. [3] is to group log messages based on their co-occurence. Log messages which frequently occur together with a short time interval are grouped together.
For the simplification step many different approaches exist. A possible approach is to build an automaton [12]. Shang et al. [8] use an easier approach, which only searches for repetitions and permutations in and between log message groups. Also a clustering based approach is possible [6].
As shown by Bezerra und Wainer [1], if a process model has been constructed, it is possible to use this process model for anomaly detection.

## 3. The Proposed Approach

The proposed approach consists of three steps: trace reconstruction, process grouping and process mining. In the trace reconstruction step, log messages that belong to a case are grouped. In order to create an approach that fulfils this task, the procedures described in the related work are collected and analysed. Procedures that have a similar structure or overlap are merged and generalized. From these no longer overlapping procedures, groups of procedures

are selected whose combinations are promising. Promising combinations are those in which the methods complement each other, but do not necessarily consider only the same features. These combinations are then implemented and evaluated. In addition, a procedure is developed which uses correlations between parameters to perform clustering.
The second step, the process grouping, is based on the method developed by Lin et al. [6]. It may be necessary to adjust the type of weighting in order to achieve a consistent result. Several attempts are made to find an optimal value for the threshold parameter $\theta$. For process mining in the third step, existing methods for which there is an available and applicable implementation are used. If none of the existing methods achieve a satisfactory result, a method will be developed which is based on simplifications such as those of Shang et al. [8]. A possible different method is the construction of automata as proposed by Yu et al. [12].
Evaluations are necessary in many of the steps to optimize parameters and for the final evaluation. To be able to perform an evaluation, patterns are searched in known system logs manually. The patterns are then grouped by type (sequences, branches, loops, ...) to compare different scenarios. In order to carry out the evaluation in practice, an environment is created for the implemented procedures, which takes over the input of parsed log data and the processing of the traces created by the procedures. The parsing and abstraction of log data is carried out manually in a first step. The use of automatic log abstraction methods like those developed by Fu et al. [4] and Jiang et al. [5] is possible.
The finished processes are then evaluated. The patterns recognized by this approach are compared with the manually extracted patterns. Special attention is paid to patterns being discovered in this step, which are previously overlooked at manual extraction. If such patterns exist, this indicates that this approach has an advantage over humans in this respect. An automated way to evaluate this approach is to use log data, which already have a case information (linux syslog messages log with the pid or hadoop logs), as a starting point, but remove the task-id for processing. Afterwards, it is compared how many of the traces could be restored from these logs. Three cases are considered for each log message: Firstly, the log message has been correctly placed in a trace, so there is a trace in the ground truth that has exactly the same log messages. Secondly, the log message has been placed in a trace that exists in the ground truth without the currently examined log message. Thirdly, the log message has been placed in a trace that does not exist in the ground truth even without the log message just examined. In addition, the Levenshtein distance can be used to measure the distance between a recognized trace and the most similar trace in the ground truth.
A comparison with the procedures described in the related work is difficult to make, since most procedures use Hadoop log data that already have a task-id and thus a case assignment. Also the goal of most approaches described in the related work is to make a problem identification and not anomaly detection, where the used metrics of both fields are not directly comparable.

## 4. Experimental Evaluation

In order to show that this approach can work, a short example was worked out using a very simple and structured log. The log data are logs collected from a DHCP server. The log data was parsed manually and the most important fields (in table 1 enclosed with < and >) were extracted. These extracted fields are referred to as the attributes of a log message. The different log message types are called event types in the following and are described with their associated codes in table 1. Each log message consists of a timestamp and the data of the corresponding event type.

When viewing the log file manually, some (not simplified) patterns can be identified very quickly:

- A D
- C B A D
- E E E
- F C B F A D
- I J K L M N O I J K L E E E H G H G H G P

| Code | Log Message |
|---|---|
| A | DHCPREQUEST for <ip> from <mac> via <interface> |
| B | DHCPOFFER on <ip> to <mac> via <interface> |
| C | DHCPDISCOVER from <mac> via <interface> |
| D | DHCPACK on <ip> to <mac> via <interface> |
| E | Wrote <num> <items> to leases file. |
| F | reuse_lease: lease age <seconds> (secs) under 25% threshold, reply with unaltered, existing lease for <ip> |
| G | Sending on <interface> |
| H | Listening on <interface> |
| I | Internet Systems Consortium DHCP Server <version> |
| J | Copyright <from>-<to> Internet Systems Consortium. |
| K | All rights reserved. |
| L | For info, please visit https://www.isc.org/software/dhcp/ |
| M | Config file: <path> |
| N | Database file: <path> |
| O | PID file: <path> |
| P | Server starting service. |

Table 1. Event-types and assigned codes

These manually recognized patterns are expected to be recognized by the automated process.

### *4.1. Procedure*

To simplify the trace reconstruction, the following assumption was made: Events, that belong to one trace occur directly after each other. Events from different traces are not interleaved in the log data. Then two consecutive events are taken into account. If they share more common than mismatching attributes, both are combined into one trace. In this way, the log file is gone through event by event and all events are placed in traces.

In this simple example, traces were only combined into a process group if the sequence of events was identical. To do this, the codes of the event types of a trace are appended one after the other and are regarded as words. These words can then be compared.

In order to simplify the third step, process mining is omitted and only an approach as in [8] is followed to simplify sequences. For this purpose the sequences were simplified based on repetitions. In contrast to [8], not only are repetitions of individual events within a trace removed, but also repetitions of pairs of two-grams of events in a trace are removed. This simplifies a sequence `ABBBCDCDCDEF` to `ABCDEF`.

### *4.2. Results*

Table 2 shows the (simplified) sequences extracted by the automatic procedure in comparison to the now simplified manually identified sequences mentioned in section 4.

The two manually recognized sequences `AD` and `CBAD` have also been recognized by the automatic procedure. There are some differences in the remaining sequences. The two sequences `ADIJKLMNOIJKLEHGPAD` and `ADIJKLMNOIJKLEHGPCBAD` are similar to the sequence `IJKLMNOIJKLEHGP`, with the difference that the two recognized sequences each contain an already known sequence (`AD` and `CBAD`) at the beginning and at the end. This clearly shows that there is still room for improvement in trace reconstruction. In the current procedure, events that actually belong to different processes and should therefore also be in different traces were erroneously combined in one trace. The biggest problem here is that consecutive events that do not have common attribute types (for example, event type `D` with the attributes ip, mac and interface and event type `I` with the attribute version), which is why both the number

| manual | automatic |
|---|---|
| A D | A D |
| | A D E A D |
| | A E D |
| C B A D | C B A D |
| | C B A E D |
| E | |
| F C B F A D | C B F A D |
| | F A D |
| | F |
| I J K L M N O I J K L E H G P | A D I J K L M N O I J K L E H G P A D |
| | A D I J K L M N O I J K L E H G P C B A D |

Table 2. Manually and automatically extracted event sequences

of attributes that are the same and the number of attributes that are different are 0. This procedure causes the conflict that events D and I do not have common attributes and should not be included in a trace together, but the events I and J belong together in a trace but do not have common attributes either. A procedure based purely on attribute values will therefore not suffice.

By simplifying sequences with directly consecutive repetitions of pairs of two events, some sequences have been generalized very well. Before the simplification there were the sequences CBCBAD, CBCBAD, CBCBCBAD, CBCBCBAD and CBAD which were all merged to the sequence CBAD by the simplification.

In addition, it is noticeable that the sequence EEE (simplified to E) occurs as an insertion in two other sequences. If in the sequence CBAED the E event would be removed, the already known sequence CBAD is created. A similar case is with the sequence ADFAD. Here the removal of the F event creates two times the already known AD sequence. These two cases are an indication that the assumption made at the beginning that events from one trace occur directly after each other and that events from different traces do not mix could be incorrect. Especially for log files written by systems that have more than one process, this assumption will not be correct. For these cases, there must be chosen a different procedure for the trace construction.

## 5. Conclusion

The evaluation has already shown that there is still room for improvement in trace reconstruction. It has been shown that the trace reconstruction cannot be performed exclusively on the correlation of attribute values, since there are cases that do not have any correlation, but belong together in part in a trace, and cases that do not have any correlation of the attribute values and do not belong together in a trace. Here, an approach such as that of Du et al. [3] with the co-occurence could provide additional information.

Further solutions must be found for logs in which different traces occur nested within each other. Alternatively, it can be attempted to detect insertions in the sequences and to remove them afterwards. Subsequences that occur at the beginning or end could also be removed in order to obtain a unique sequence.

## References

[1] Bezerra, F., Wainer, J., 2008. Anomaly detection algorithms in logs of process aware systems, in: Proceedings of the 2008 ACM Symposium on Applied Computing - SAC '08. Presented at the the 2008 ACM symposium, ACM Press, Fortaleza, Ceara, Brazil, p. 951. https://doi.org/10.1145/1363686.1363904

[2] Breier, J., Branišová, J., 2015. Anomaly Detection from Log Files Using Data Mining Techniques, in: Kim, K.J. (Ed.), Information Science and Applications, Lecture Notes in Electrical Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 449–457. https://doi.org/10.1007/978-3-662-46578-3_53

[3] Du, M., Li, F., Zheng, G., Srikumar, V., 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Presented at the CCS '17: 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, Dallas Texas USA, pp. 1285–1298. https://doi.org/10.1145/3133956.3134015

[4] Fu, Q., Lou, J.-G., Wang, Y., Li, J., 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis, in: 2009 Ninth IEEE International Conference on Data Mining. Presented at the 2009 Ninth IEEE International Conference on Data Mining (ICDM), IEEE, Miami Beach, FL, USA, pp. 149–158. https://doi.org/10.1109/ICDM.2009.60

[5] Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P., 2008. An automated approach for abstracting execution logs to execution events. J. Softw. Maint. Evol.: Res. Pract. 20, 249–267. https://doi.org/10.1002/smr.374

[6] Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., Chen, X., 2016. Log clustering based problem identification for online service systems, in: Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16. Presented at the the 38th International Conference, ACM Press, Austin, Texas, pp. 102–111. https://doi.org/10.1145/2889160.2889232

[7] Lou, J.-G., Fu, Q., YANG, S., XU, Y., Li, J., 2010. Mining Invariants from Console Logs for System Problem Detection, in: Annual Technical Conference (Full Paper). USENIX.

[8] Shang, W., Jiang, Z.M., Hemmati, H., Adams, B., Hassan, A.E., Martin, P., 2013. Assisting developers of Big Data Analytics Applications when deploying on Hadoop clouds, in: 2013 35th International Conference on Software Engineering (ICSE). Presented at the 2013 35th International Conference on Software Engineering (ICSE), IEEE, San Francisco, CA, USA, pp. 402–411. https://doi.org/10.1109/ICSE.2013.6606586

[9] van der Aalst, W.M.P., de Medeiros, A.K.A., 2005. Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. Electronic Notes in Theoretical Computer Science 121, 3–21. https://doi.org/10.1016/j.entcs.2004.10.013

[10] van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M., 2003. Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47, 237–267. https://doi.org/10.1016/S0169-023X(03)00066-1

[11] Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I., 2009. Detecting large-scale system problems by mining console logs, in: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles - SOSP '09. Presented at the the ACM SIGOPS 22nd symposium, ACM Press, Big Sky, Montana, USA, p. 117. https://doi.org/10.1145/1629575.1629587

[12] Yu, X., Joshi, P., Xu, J., Jin, G., Zhang, H., Jiang, G., 2016. CloudSeer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs. SIGARCH Comput. Archit. News 44, 489–502. https://doi.org/10.1145/2980024.2872407