Discovering Cloud Operation History through Log Analysis

Mitsunari Kobayashi Yosuke Himura Yoshiko Yasuda
Research & Development Group, Hitachi, Ltd.

***Abstract*—Management costs in private clouds will be promisingly reduced by reviewing 'operation history,' which is defined as a holistic view of past operation executions. Operation history provides insights into breakdown of operations: the breakdown clarifies cost-dominant operations to be improved and repetitive ones to be automated. Towards obtaining the operation history, a conventional approach relying on manual investigation is time-consuming, and another relying on agent-based monitoring is not often acceptable in sensitive mission-critical enterprise clouds. Different from these approaches, our idea is to discover the operation history by automatically analyzing 'system logs' that are easily accessible even in sensitive clouds. Since system logs contain only low-level debugging messages about programmatic events without direct contexts about operations, the challenge is to recover high-level operational contexts from low-level system logs. To address this challenge, we develop a method that first abstracts system logs using a pre-defined event sequence model, and then maps the abstracted events to high-level individual operations—this mapping between different contextual levels is achieved by using complementary cross-cloud reference data. Evaluation of an implementation revealed that this method reduces the time taken to discover the history by 99.9% compared to a conventional approach while achieving up to 95% correctness.**

Keywords—private cloud; cloud management; operation; automation; business process management

1. INTRODUCTION

Private clouds, which enable datacenter IT resources (e.g., servers, networks, storage devices) to be used in a virtualized manner, have been adopted by many enterprises. One motivating advantage of private clouds is that IT systems, which have traditionally been built over separated sets of physical devices, can be consolidated into a smaller number of physical devices that are shared yet logically separated by virtualization. Such consolidation reduces device costs [1].

On the other hand, the adoption of private clouds increases management costs. This is because the use of private clouds requires an additional management layer (a virtualization layer), that includes such operations as deploying new virtual machines (VMs) and setting up virtual networks. In many cases, these operations are done manually (e.g., by using GUI-based cloud management tools, such as VMware®[1] vSphere Web Client and OpenStack®[2] Horizon) by operators. These manual operations consume a large amount of management time, which could be reduced by, for instance, automating them.

The time needed for these operations could be reduced by making use of 'operation history,' which we define as a holistic view of past operation executions. The operation history provides information about when and what types of operations were performed over which IT resources. Its use would provide insights into the time spent on the additional management operations for the virtualized resources. For example, automation of the operations taking the most time and of the ones repeatedly executed could be prioritized.

However, conventional approaches to obtaining operation history are time-consuming or not widely applicable. A representative conventional approach relies on manual investigation of handwritten operational records as well as visually observing the behavior of operators while they perform operations; this approach is thus time-consuming. Another approach is to automatically monitor terminal devices where the operators perform various operations for cloud management by using additionally installed agent software (e.g., VMware®[1] Onyx); however, the installation of additional tools in sensitive environments (e.g., mission-critical enterprise clouds) is not generally acceptable.

Given these problems in obtaining the operation history, we took a different approach: analyze log data that are machine-readable as well as natively accumulated in a private cloud by default. In a preliminary study, we found that there are two types of logs: *system logs* and *configuration change records*. The former contain arbitrary types of low-level debugging messages recorded by programmatic events executed in clouds without direct operational contexts (i.e., not interpretable) although they are easily and immediately retrievable. The latter contain information about the context of operation executions and corresponding low-level programmatic events, which were triggered by the operation in the clouds, in the form of structured database records (i.e., interpretable), so they are useful for obtaining operation history. However, since the configuration change records are usually stored in production databases on sensitive mission-critical enterprise clouds, retrieving a number of records may not be necessarily or promptly acceptable due to concerns such as overloading the databases. Given these findings, we focused on using system logs to discover operation history; the challenge is thus to recover high-level operational contexts from low-level debugging messages.

To meet this challenge, we developed a method for discovering operation history from system logs. It first extracts information possibly related to past operations and then

[1]VMware is a registered trademark of VMware, Inc.
[2]OpenStack is trademark of the OpenStack Foundation.

maps it to the high-level operational context. This recovery of operational contexts is realized through cross-cloud complementary analysis—the system logs in a mission-critical enterprise clouds (the operation history of which is to be obtained) are analyzed using cross-cloud reference data containing the relationships between low-level events and high-level operational contexts. These reference data are constructed through pre-investigation over data sources (e.g., configuration change records) obtained from another cloud that is not mission-critical (e.g., an existing experimental cloud). The originalities of this contribution include the basic approach of discovering operation history from system logs and the method we developed.

2. Features of System Logs and Challenges

As mentioned above, our approach is to automatically discover operational history through system log analysis. In this section, we describe the features of system logs (Sec. 2.1), introduce our data model for the operation history (Sec. 2.2), and discuss the technical challenges (Sec. 2.3).

2.1. Features of system logs

System logs are generally the data accumulated for the text messages generated in a computer system. They are characterized by two features: unformatted text messages and low-level event records.

***Feature 1**: Unformatted text messages*

The data in a system log is typically recorded as unformatted text messages, as shown in Fig. 1. These messages are usually written in natural language plain text, which does not have a data schema or parameters with a direct meaning.

***Feature 2**: Low-level event records*

The text messages are recorded in accordance with the execution of low-level events by arbitrary programmatic behavior; the corresponding high-level operational contexts are not recorded. Once an operation is performed by an operator, the cloud management software breaks down the operation into a sequence of low-level programmatic events, as shown in Fig. 2. These events are recorded without the corresponding high-level operational contexts or the interrelations among events triggered by the same original operation.

```
ection for cloud cloudManager, pid=4695, version=5.1.0, build=1123961, option=Release
2016-02-13T11:38:38.778Z [7FA1C84C3700 info 'vpxdvpxdVmomi' opID=SWI-84fd8e7d]
[ClientAdapterBase::InvokeOnSoap] Invoke done (host.test.domain.com,
vpxapi.VpxaService.fetchQuickStats)
2016-02-13T11:38:39.929Z [7FA1C84C3700 info 'vpxdvpxdVmomi' opID=SWI-84fd8e7d]
[ClientAdapterBase::InvokeOnSoap] Invoke done (host.test.domain.com,
vpxapi.VpxaService.fetchQuickStats)
2016-02-13T11:38:39.930Z [7FA1C84C3700 info 'vpxdvpxdVmomi' opID=SWI-84fd8e7d]
[ClientAdapterBase::InvokeOnSoap] Invoke done (host.test.domain.com,
vpxapi.VpxaService.fetchQuickStats)
2016-02-13T11:39:18.737Z [7FA1C8135700 info 'commonvpxLro' opID=6cf9d0c7]
[VpxLRO] -- BEGIN task-internal-2624719 --  -- vim.event.EventManager.GetLatestEvent -
- 8c7701a9-f96b-c382-21ec-f69750749c55(52142d9e-2201-e9d6-1bb2-3013ddb81002)
2016-02-13T11:39:18.738Z [7FA1C8135700 info 'commonvpxLro' opID=6cf9d0c7]
[VpxLRO] -- FINISH task-internal-2624719 --  -- vim.event.EventManager.GetLatestEvent
2016-02-13T11:39:29.256Z [7FA1C88EA700 info 'commonvpxLro'
2016-02-13T11:39:18.738Z [7FA1C8135700 info 'commonvpxLro' opID=6cf9d0c7]
```

Fig. 1 Example contents of system log

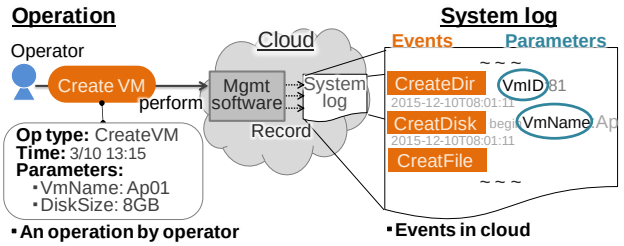

Fig. 2 Breakdown of an operation into multiple events

2.2. Data model for operation history

We formally define the operation history as $\{O_1, O_2, O_3, .., O_n\}$, with $O_i = \{opType_i, time_i, r_i, \boldsymbol{p}_i\}$, where *opType* is the type of operation (e.g., CreateVM), *time* is the time the operation was performed, r is the ID of the resources over which the operation was conducted (e.g., VM ID), and $\boldsymbol{p}$ is a set of arbitrary parameters (e.g., configuration settings such as disk size).

This data model can also be used for visualization purposes, as shown in Fig. 3(a). The individual operations are aligned by resource ID and sorted by time. The detailed attributes of the individual operations can be further visualized through key-value representation, as shown in Fig. 3(b). Such visualization provides a holistic view of operations (e.g., time-consuming operations and/or repetitive operations) as well as a drill-down view of individual operations.

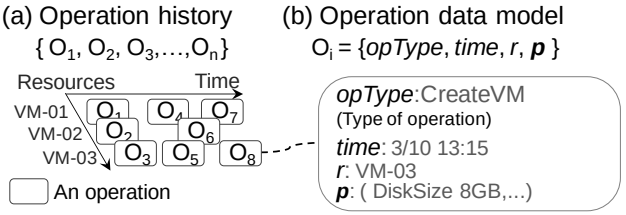

Fig. 3 Operation history and operation data model

2.3. Technical challenges

Discovering the operation history from system logs poses two technical challenges corresponding to the two features of system logs.

***Challenge 1**: Analyzing unformatted system logs*

The first challenge is dealing with the unformatted nature of system logs since unformatted texts are not well suited for recovering information about various types of operations and events. We therefore first need to extract the semantics of the operation-related meanings in the text messages and then abstract them as common representations that are suitable for recovering the operation history.

***Challenge 2**: Recovery of operational context*

The other challenge is to recover information about individual operations, especially operation types, from low-level events, in which the corresponding high-level operational context were not directly recorded. This lack of context is the main problem characterizing the present work. We have to compensate for this contextual gap between the high-level context of operation types and the low-level events.

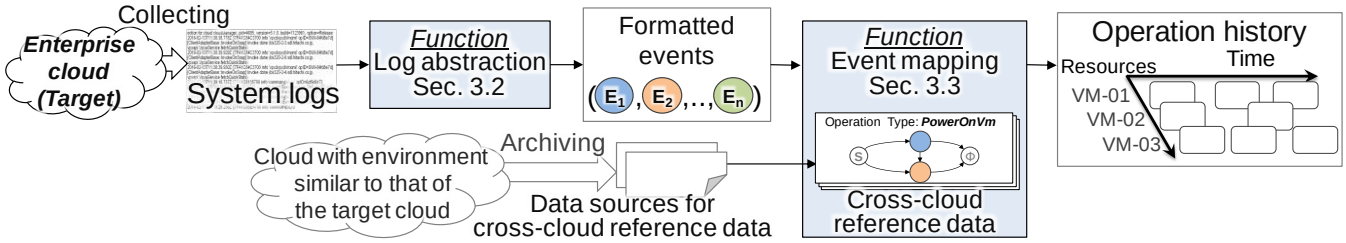

Fig. 4 Overview of proposed method

3. DEVELOPED METHOD

The method we developed for discovering operation history from system logs addresses the two challenges described in the previous section. In this section we first overview the developed method (Sec. 3.1) and then describe its two functions (Secs. 3.2 and 3.3).

3.1. Overview

From a high-level viewpoint, operation history is discovered from the system log generated in a sensitive mission-critical enterprise cloud, the operation history of which is to be investigated. As shown in Fig. 4, system logs are processed using two functions: "log abstraction" (to meet Challenge 1) and "event mapping" (to meet Challenge 2). The log abstraction function is used to abstract the unformatted system log into machine-readable representations of the event sequence. The event mapping function is used to reversibly map the individual sub-sequences of the abstracted event sequence to the corresponding operations by using cross-cloud reference data. These reference data are constructed with this function by using different data sources (e.g., configuration change records [3]) in another less-sensitive cloud (e.g., experimental cloud) the internal software stack of which is similar (and compatible) to that of the sensitive target cloud. This cross-cloud reference data helps to compensate for the lack of context of event-operation relationships. This cross-cloud complementary analysis should be feasible as long as both cloud environments use the same management software.

3.2. Log abstraction function

This function transforms unformatted text messages in a system log into a machine-readable event sequence. Although the text-pattern matching (mentioned later) in this function can basically be accomplished by applying corresponding part of existing work such as Ref.[2] and is not the core of the present work's originality, we here define a data model tailored for the challenges and describe a practical example of the abstraction process in order to make this paper self-contained.

[3]The present work uses configuration change records as a representative data source under the assumption that such a data source exists. Another referential data source may be obtained with a black-box-like method that examines individual output from intentionally executing all types of operations in an experimental cloud. Since such an active-probing approach would not be widely acceptable in production environments, the spirit of cross-cloud analysis approach used in the present work will also apply. This method is free from that assumption but requires human labor.

1) Definition of event sequence data model

Since a system log can be regarded as an accumulation of text messages about programmatic events, as mentioned above, we define the data model as a sequential set of events $(\mathrm{E}_1, .., \mathrm{E}_n)$, with $\mathrm{E}_i = \{\mathrm{ET}_i, time_i, \boldsymbol{p}_i\}$. In this model, *ET* is the event type (e.g., "CreateDisk"), *time* is the event execution time, and $\boldsymbol{p}$ is a list of parameters related to the event (found in text massages).

This function abstracts system logs by parsing each text message line and extracting parameters for the data model by using conventional text-pattern matching with a predefined collection of regular expressions. For example, suppose that a line of text in the system log is '2016-11-30T01:10:33 BEGIN CreateDisk vm-id vm-01,' and there is corresponding regular expression '*(*1) BEGIN (*2) vm-id (*3)*.' The line with the matching parameters is abstracted as E={*time*=(*1),ET=(*2), $\boldsymbol{p}$=[vmId=(*3)]]}. The E represents $\{CreatedDisk, 2016/11/30\mathrm{T}01{:}10{:}33, [\mathrm{vm}-01]\}$. Processing of the entire system log results in the corresponding data model representation: $\boldsymbol{E} = (\mathrm{E}_1, \mathrm{E}_2, .., \mathrm{E}_n)$.

3.3. Event mapping function

This function discovers individual operations from the abstracted event sequence. Since there are no direct relationships between the operations and events in the system log, we first need to consider how the relationships can be recovered.

1) Design consideration

Cross-cloud analysis using cross-cloud reference data obtained from a different cloud. As mentioned in Sec. 1, cloud environments accumulate data sources (e.g., configuration change records) containing information about past operations and their relationships to events, but these data in the target cloud (e.g., a mission-critical enterprise cloud) are not always easily accessible. Our basic idea for overcoming this problem is to utilize the information from another cloud (e.g., an experimental cloud) that is not mission critical as cross-cloud reference data for analyzing the system log of the target cloud.

Model of cross-cloud reference data. We formally define each referential context regarding operation type and sequence of event types:

$$Context = \{opType, (\mathrm{ET}_a \rightarrow \mathrm{ET}_b \rightarrow, ..)\},$$

where *opType* is the operation type, $\boldsymbol{ET} = (\mathrm{ET}_a \rightarrow \mathrm{ET}_b \rightarrow, ..)$ is the sequence of event types, and ET_x represents a specific event type (e.g., $\mathrm{ET}_{\boldsymbol{CreateDisk}}$).

An example of configuration change records is shown in Fig. 5(a). Each row has "Event sequence" and "Operation type," which can be regarded as the context between events and operations. Since some contexts with an identical operation type can have a variation in the event sequences, and observed records represent only individual cases (not a general case), these sequences are aggregated into the canonical form of a directed graph G_{opType}:

$$G_{opType} = \{Vertex_{opType}, Edge_{opType}\},$$
$$Vertex_{opType} = \{S, ET_a, ET_b, .., \phi\},$$
$$Edge_{opType} = \{(S \rightarrow ET_*), .., (ET_* \rightarrow \phi)\},$$

where S is the start of the event sequence, ET_* is an event type observed in the aggregated event sequence, and ϕ is the end of the sequence. The collection of G_{opType} composes the cross-cloud reference data.

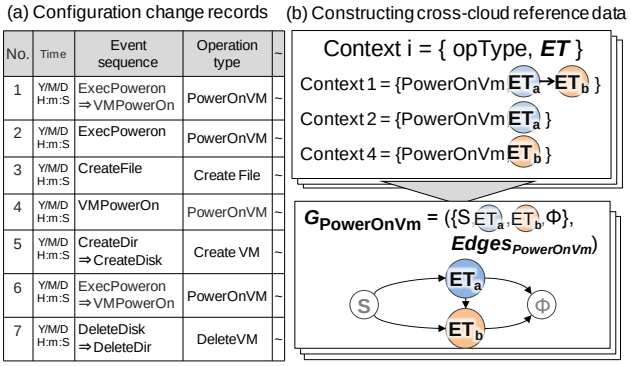

Fig. 5 Configuration change records and cross-cloud reference data

2) Pseudocode and example of event mapping

In this section, we describe how the event mapping procedure is performed using cross-cloud reference data and an example case.

Pseudocode. As shown in the following pseudocode (Fig. 6), the input data is the abstracted event sequence $\boldsymbol{E}$ and cross-cloud reference data, which is a list of G_{opType} instances, and the output data is operation history. At the beginning of the procedure, the abstracted events in $\boldsymbol{E}$ are grouped by parameter (e.g., resource ID) in $\boldsymbol{p}$ (line 1), and the groups are iteratively processed (lines 2–16). Events in a grouped event sequence ($\boldsymbol{E'}$) are iteratively mapped to the corresponding operations by performing pattern matching between the sub-sequences in $\boldsymbol{E'}$ and all direct graph G_{opType} instances in the cross-cloud reference data (lines 3–16). In the pattern matching process, each event type sequence ($\boldsymbol{ET'}$) in $\boldsymbol{E'}$ is compared to the event patterns defined on G_{opType} (lines 4–12). If a matching pattern is detected, the sub-sequence is regarded as the set of events triggered by the same operation. The matched sub-sequence is extracted from $\boldsymbol{E'}$ and stored as an operation after merging values in the $\boldsymbol{p}$ of the events and mapping the G_{opType} operation type (lines 11–16).

Example case. In the example case shown in Fig. 7, the abstracted events are already grouped by parameter (such as by "VM-03"), and one of the grouped event sequences is shown as $\boldsymbol{E'}$. Pattern matching is performed between the sub-sequences in $\boldsymbol{E'}$ and $G_{PowerOnVm}$. In this example, matching sub-sequences (ET_a) and (ET_a, ET_b)are detected, so the detected sub-sequences are mapped to the operation type "PowerOnVm" and stored as operations in the data model shown in Fig. 3(b).

Pseudocode: *Event mapping procedure*

Input: *Event sequence*: $\boldsymbol{E} = (E_1, .., E_n)$,
where $E_i = \{ET, time, \boldsymbol{p}\}$
Cross-cloud reference data: $\{G_{opType}, ..\}$
Output: *Operation history*: $\{O_0, O_1, .., O_n\}$

```
1  group events in E by resource ID in p of E_i
2  for each grouped event sequence E':
3      for each G_opType
4          get event type seq. ET' = (ET_0', .., ET_j') of E'
5          current ET = ET_0', current Vertex = S in G_opType
6          while current Vertex ≠ ϕ
7              if current ET exists in next Vertex in G_opType
8                  current ET = next ET', current Vertex = next ET'
9              else if next Vertex = ϕ
10                 current Vertex = ϕ
11             else
12                 break
13         if current Vertex = ϕ
14             extract E' from E and merge p of E in E'
15             map opType of G_opType onto E'
16             store E' with opType as an operation O
```

Fig. 6 Pseudocode for event mapping procedure

4. EVALUATION

We evaluated an implementation of our method in terms of processing time (Sec. 4.1) and feasibility of correctly discovering past operations by using cross-cloud reference data (Sec. 4.2).

4.1. Processing time

Motivation. As stated in Sec. 1, the conventional approach to obtaining operation history relies on time-consuming manual investigation; this approach can take several days and much human labor. Hence, it is important to evaluate the reduction in human labor to evaluate the effectiveness of developed method

Evaluation method. We estimated the processing times of the conventional and developed methods as follows.

- **Conventional.** We considered a practical case in which the aim is to improve VM deployment operations in a private cloud consisting of hundreds of VMs in a multi-tenant manner. Conventionally, the investigation procedure is composed of such phases as interviewing operators and checking paper-based operational documents. We estimated the time taken for each phase by interviewing the investigators involved in the sample case.
- **Developed.** The processing time of the developed method is mostly determined by the computation (rather than I/O) time. We evaluated this time for about 10 GB of system logs, with tens of millions lines in total, by using an ordinary personal computer with a 2.50-GHz two-core CPU, 4 GB of memory, and a 64-bit OS.

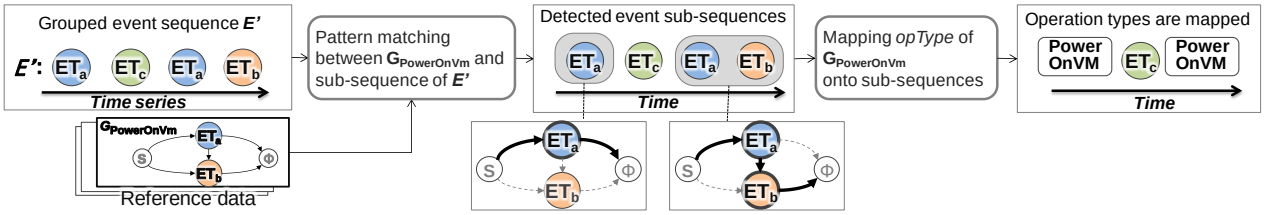

Fig. 7 Detecting event subsets and mapping operation types according to the reference data

Results. As shown in Table 1 it took more than seven days with the conventional method and about 15 minutes with the developed method, a reduction of 99.9%.

Table 1 Time taken to discover operation history

	Phased procedure	Conventional	Developed method
1	Interview operators	4.0 days	-
2	Check operation manuals	1.0 days	-
3	Investigate paper-based operation records	5.0 days	-
4	Analyze system log	-	15 min
	Sum	**10.0 days**	**15 min**
	Percent reduction	**-**	**99.894%**

Table 2 Mapping accuracy on cross-validation evaluations

	Evaluation *Analyzing target cloud's system log with reference data from another cloud*	Target: Cloud A Ref. Cloud B	Target: Cloud B Ref. Cloud A
1	(i) no. of operations in the system log	4212	2317
2	no. of unmapped operation	1263	110
	2-1: Lack of operation type in reference data	59	42
	2-2: lack of event sequence in reference data	1204	68
3	no. of incorrectly mapped operations	0	12
4	(ii) no. of correctly mapped operations	2949	2205
5	% of correctly discovered operations in the system log	70.01	95.16

4.2. Feasibility of correctly discovering operations

Motivation. The cross-cloud complementary analysis is based on the assumption that the event sequences for individual operations are not much different between clouds, in addition to the assumption that these clouds use the same management software. Hence, it is important to evaluate the correctness of the recovered contexts in the event mapping function in order to validate this assumption, which is the core of the developed method.

Evaluation method. We evaluated the correctness by comparing (i) operations recorded in the configuration change records as a supervisory dataset and (ii) operations recovered from system logs, in a cross-validation manner between two experimental clouds (A and B). We first performed reverse-mapping of the system log from Cloud A with the cross-cloud reference data from Cloud B and then compared the mapping results to the configuration change records from Cloud A. We then did the same procedure starting from Cloud B. The clouds used almost the same versions of cloud management software from VMware®.

Results. As shown in Table 2, the system logs from each cloud contained more than 2000 operations each (Row 1). The numbers of mapping errors are shown in Rows 2 and 3. There were two types of mapping error: unmapped operations and incorrectly mapped operations. Row 5 shows the overall accuracy, i.e., the fraction of operations mapped accurately—more than 70% of the operations were correctly discovered.

Row 2 further breaks down the number of unmapped operations by cause.

- Row 2-1 shows the number of operations unmapped due to the corresponding operation type not being included in the cross-cloud reference data. In a practical sense, those types of operations were executed in Cloud A and not in Cloud B, resulting in the lack of corresponding reference information. Examples of such unmapped types of operations included taking a snapshot of the VM image and disconnecting a virtualized host server. The number of such unmapped operations was less than 100 out of 1263 for both clouds.
- Row 2-2 shows the number of unmapped operations due to unrecorded variations in event sequences for certain operation types. This variability caused difficulty in building always applicable referential information for those operation types. For example, as shown in Fig. 7, which shows data for Cloud A, the operation type "PowerOnVM" triggered three event sequence patterns: (ET_a, ET_b), (ET_a), and (ET_b). However, the sequence (ET_a, ET_b) was not detected in Cloud B; on the other hand, we found that the operation type "PowerOnVM" in Cloud B triggered only (ET_a) or (ET_b). Such differences in event triggering would result in unmapped operations. In the case of Cloud A, there were 1204 operations unmapped due to such differences.

For the incorrectly mapped cases, additional investigation revealed that there were situations in which different types of operations (e.g., "remove disk image" and "remove a configuration file") triggered the same programmatic event (e.g., remove a file). This is reasonable because the cloud management software executes the same operational program for different operations. Precise reverse-mapping of those types of operations is not an easy task.

5. DISCUSSION

Practical uses. The evaluation revealed that the mapping correctness ranged from 70% to 95%. Although this level of correctness is not 100%, it is sufficient for detecting repeatedly executed and time-consuming operations and for identifying specific operations to be improved. The level of correctness could be increased by manual investigation with less overhead than for the conventional method.

Limitations. One limitation of our approach is the limited coverage of the cross-cloud reference data (Sec. 3.3). Since operation types are mapped to events in accordance with the

cross-cloud reference data, the level of mapping correctness greatly depends on the information richness of the source of the cross-cloud reference data. The level can thus be improved by increasing the volume of the data source, and this is not much difficult because it can be achieved by merely concatenating other data sources from various cloud environments that adapt similar cloud management software.

Towards improving mapping correctness. As mentioned above and shown in Table 2, the cross-cloud reference data affects level of mapping correctness (i.e., the no. of unmapped cases and the no. of incorrectly mapped cases). Possible future work includes combining data sources for the cross-cloud reference data generated from more than one environment (to reduce the number of unmapped cases).

6. RELATED WORK

Unstructured log analysis. Analysis on unstructured log data has been performed by many researchers. Most aimed at detecting anomalies in a system. Reidemeister et al., for example, analyzed log data for classifying recurrent faults in software in order to facilitate fault diagnosis [3]. Other researchers have tried to proactively capture abnormal behaviors by detecting specific patterns appearing in log data generated in a virtualized IT infrastructure [4], server [5], network [6], or distributed system [7]. Although their approaches and ours are similar in that useful information is extracted from system logs (e.g., using pattern matching with pre-defined log pattern templates), we do not share the goal with them. Whereas their goals are to detect abnormal events or event patterns by classifying the extracted information, our goal is to recover operational contexts hidden in the extracted events in system logs.

Various researchers have worked on log abstraction, and several tools have been developed that can be complementarily used with our developed method. For example, a text mining approach automatically generates regular expressions for log abstraction [2]. This corresponds to the collection of pre-defined regular expressions in the developed method (pre-processing phase). Furthermore, conventional log management tools, such as Logstash®[4], perform pattern matching with pre-defined regular expressions (in addition to basic functionalities such as log collection and storage). These tools can be applied in pre-processing phase in the developed method.

Business process discovery. A number of researchers have focused on discovering business process workflows. This has been done, for example, by analyzing logs of individual approval events stored in a well-structured relational database of business applications [8][9]. Whereas these prior efforts and ours share a similar spirit of recovering processes from log data, the differences are derived from the goals (i.e., approval workflow and operational history in ours) and the data analyzed in their approaches (i.e., event logs of business applications vs. cloud system logs). The difference in the data used motivated us to develop an original method for analyzing unstructured system logs containing low-level events to recover contexts of operational processes. At the view of discovering cloud operations, Yanase et al. developed a method for extracting operations described in document data of manual books by using natural language processing [10]. However, the extracted operations do not contain information about past operation executions and thus will not give insights into improving cloud management such as automating repeatedly performed operations as operation history does.

[4]Logstash is a registered trademark of Elasticsearch BV.

7. CONCLUDING REMARKS

We have tackled the challenge of efficiently obtaining operation history. Our basic idea is to automatically discover operation history by analyzing system logs, which indirectly involve information about past operation executions. We developed a method for extracting useful information about past operations hidden in unstructured system logs, which record low-level programmatic events. The system logs are first abstracted using a pre-defined event sequence model, and then the low-level abstracted events are mapped to high-level individual operations—this mapping between different contextual levels is achieved by using complementary cross-cloud reference data. Evaluation of an implementation revealed that our approach reduced the time taken to obtain operation history by 99.9% compared to a conventional approach while achieving up to 95% correctness. Future work includes increasing mapping correctness by extending the amount of cross-cloud reference data.

REFERENCES

[1] B. Sotomayor, R. S. Montero, I. M. Llorente and I. Foster, "Virtual Infrastracture Management in Private and Hybrid Clouds," IEEE Internet Computing, 2009.

[2] S. Kobayashi, K. Fukuda and H. Esaki, "Towards an NLP-based log template generation algorithm for system log analysis," ACM, CFI, 2014.

[3] T. Reidemeister, M. Jiang and P. A. Ward, "Mining Unstructured Log Files for Reccurent Fault Diagnosis," IFIP/IEEE, IM, 2011.

[4] M. A. Marvasti, A. V. Poghosyan, A. N. Harutyunyan and N. M. Grigoryan, "Pattern detection in unstructured data: An experience for a virtualized IT infrastructure," IFIP/IEEE, IM, 2013.

[5] R. Vaarandi, "A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs," Springer, Intelligence in Communication Systems, 2004.

[6] A. N. Harutyunyan, A. V. Poghosyan, N. M. Grigoryan and M. A. Marvasti, "Abnormality Analysis of Streamed Log Data," IEEE, Network Operations, 2014.

[7] Q. Fu, J.-G. Lou, Y. Wang and J. Li, "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis," IEEE, ICDM, 2009.

[8] R. Agrawal, D. Gunopulos and F. Leymann, "Mining Process Models from Workflow Logs," EDBT, Advances in Database Technology, 1998.

[9] T. A. Weijiters and W. M. van der Aalst, "Process Mining Discovering Workflow Models from Event-Based Data," BNAIC, 2001.

[10] T. Yanase, M. Asaoka, S. Onodera and I. Namba, "Assessment Method of Operational Procedure for Runbook Automation," IFIP/IEEE, IM, 2015.