An Experience Report of Generating Load Tests Using Log-recovered Workloads at Varying Granularities of User Behaviour

Jinfu Chen†, Weiyi Shang†, Ahmed E. Hassan‡, Yong Wang◇, Jiangbin Lin◇
Department of Computer Science and Software Engineering, Concordia University, Montreál, Canada†
School of Computing, Queen's University, Kingston, Canada ‡
Alibaba Group, Hangzhou, China ◇
{fu_chen, shang}@encs.concordia.ca†, ahmed@cs.queensu.ca‡, {wangyong.wy, jiangbin.lin}@alibaba-inc.com◇

***Abstract*—Designing field-representative load tests is an essential step for the quality assurance of large-scale systems. Practitioners may capture user behaviour at different levels of granularity. A coarse-grained load test may miss detailed user behaviour, leading to a non-representative load test; while an extremely fine-grained load test would simply replay user actions step by step, leading to load tests that are costly to develop, execute and maintain. Workload recovery is at core of these load tests. Prior research often captures the workload as the frequency of user actions. However, there exists much valuable information in the context and sequences of user actions. Such richer information would ensure that the load tests that leverage such workloads are more field-representative. In this experience paper, we study the use of different granularities of user behaviour, i.e., basic user actions, basic user actions with contextual information and user action sequences with contextual information, when recovering workloads for use in the load testing of large-scale systems. We propose three approaches that are based on the three granularities of user behaviour and evaluate our approaches on four subject systems, namely Apache James, OpenMRS, Google Borg, and an ultra-large-scale industrial system (SA) from Alibaba. Our results show that our approach that is based on user action sequences with contextual information outperforms the other two approaches and can generate more representative load tests with similar throughput and CPU usage to the original field workload (i.e., mostly statistically insignificant or with small/trivial effect sizes). Such representative load tests are generated only based on a small number of clusters of users, leading to a low cost of conducting/maintaining such tests. Finally, we demonstrate that our approaches can detect injected users in the original field workloads with high precision and recall. Our paper demonstrates the importance of user action sequences with contextual information in the workload recovery of large-scale systems.**

***Index Terms*—Workload recovery, Load tests, Software log analysis, Software performance**

I. Introduction

Large-scale software systems (e.g., Amazon AWS and Googles Gmail) have brought a significant influence on the daily lives of billions of users worldwide. For example, Netflix services 150 million subscribers across the globe [1]. As a result, the quality of such systems is extremely important. Failures of such planet-scale systems can result in negative reputational and monetary consequences [2], [3]. Quite often failures in such systems are load and performance-related rather than due to functional bugs [4].

Hence, load tests are widely used in practice to ensure the quality of such systems under load. The goal of a load test is to ensure that a system performs well in production under a realistic field workload. Therefore, one must first recover a workload [5], [6] then design a load test based on the recovered workload [7]–[11].

The recovery of a field-representative workload is a challenging task. In particular, one must achieve a balance between the level of granularity of the workload and the cost to conduct a load test with such a workload. All too often, in practice, the recovered workloads are too coarse, i.e., over simplified workloads. For example, the SPECweb96 Benchmark defines a workload that only specifies the probability of accessing files such as "files less than 1KB account for 35% of all requests" [12]. Such coarse-grained workloads fail to capture the variance of user behaviour, leading to non-representative load tests.

On the other extreme, a workload can simply replay the exact field workload step by step. Although, such a workload replicates the exact user behaviour, conducting a load test using such a workload and maintaining such a workload are extremely costly. First of all, due to the large amount of users of these systems, replaying the exact workload requires the load tests to simulate every user with a great amount of contextual information and complexity. One would also need to develop simulation code for each specific sequence of events. In addition, since it is almost impossible to observe the exact same workload twice, one would constantly need to update such a detailed workload.

To reach a desirable level of granularity for a workload, prior work often clusters user bahaviour based on important aspects in the workload [11], [13], [14]. With the clusters of users, instead of maintaining millions or billions of user profiles, a workload is designed based on representative user behaviours from a considerably smaller number of clusters. For example, a recent workload clustering approach clusters users by the frequency of different user actions [11]. However, due to the high variability of users in ultra-large-scale software

systems, we argue that solely considering the frequency of actions is too coarse; instead the sequence and the context of user actions can make workloads much more representative to the actual field. Consider the following example: one user repetitively reads small pieces of data from a file then writes each of the small pieces back to the file; while another user interactively reads and writes a large number of small pieces of data to a file. A workload should capture both users differently. However, only considering the frequency of actions (read and write) would not differentiate the workloads of these two users. Adding more detailed information about these user actions would lead to a finer granularity of workload which in turn might be too costly to recover, execute and maintain.

Therefore, in this paper, we report on our experience in understanding the impact of adding different levels of details in the recovered workloads for load tests. We first replicate a prior approach that captures the frequency of basic actions [11] (we refer to this approach as *Action*). Afterwards, we design an approach that enriches the basic user actions with their context values (we refer to this approach as *ActionContext*). Finally, we design an approach that augments *ActionContext* with the frequently-appearing sequences of actions (we refer to this approach as *ActionSequence*). The three approaches use the frequency of actions, the frequency of enriched actions and the frequency of sequences of enriched actions, respectively, as signatures for each user, then group the users into clusters. Afterwards, we automatically generate load tests based the signature of the representative (center) user in each cluster.

Our study is performed on two open source systems: Apache James and OpenMRS, and two commercial systems: Google Borg and an ultra-large-scale software system from Alibaba (we refer to it as SA in the rest of the paper). We compare our three approaches by recovering workloads based on the execution logs from the subject systems, generating load tests, and running the automatically generated load tests on the subject systems. In particular, we answer these two research questions:

RQ1: **How field-representative are our generated workloads?**
ActionSequence generates the most field-representative workloads. When conducting load tests using *ActionSequence*, the throughput of 10 out of 14 user actions as well as the CPU usage are either statistically insignificant to the original workload or differ with a small/trivial effect size.

RQ2: **How many clusters of users are captured by each of our recovery workload approaches?**
The number of clusters of users is not overwhelming. The most field-representative workload *ActionSequence* is based on eight to 39 clusters of users, which is only three to six clusters more than a less field-representative workload *ActionContext*. The least field-representative workload *Action* is based on as few as two clusters of users.

The rest of the paper is organized as follows: Section II discusses the background and the related work to this paper. Section III describes our approaches in detail. Section IV presents our case study setup. Section V presents our case study results by answering our two abovementioned research questions. Section VI discusses other usage scenarios for our approaches. Section VIII discusses Section VII discusses the challenges that lessons learned from the industrial evaluation. Finally, Section IX concludes the paper.

II. Background and related work

Workload recovery is an essential part in the performance assurance of large-scale systems. Prior research proposes approaches for recovering workloads to assist in the design of load tests [15], validating whether load tests are field-representative as production [11], optimizing system performance [16], [17] and detecting system performance issues [11], [13], [14], [18], [19]. All above prior work illustrates the value and importance of recovering representative workloads.

Prior approaches for recovering and replaying workloads can be categorized along the granularity of the captured user actions. One may choose a coarse-granularity by recovering only the type of workload from a system, or to the other extreme, considering each individual user and replaying their individual workload one by one. One may anonymize all high-level user behaviours and only consider the physical metrics such as CPU [13], [14], I/O [20]–[23] and other system resources [24]. One may choose a finer granularity by building complex models such as Hidden Markov Models [21] to capture the details for each user. A pilot study by Cohen et al. [14] demonstrates that grouping workloads into a smaller number of clusters outperforms having one unified workload. Intuitively, recovering a workload at a too fine or too coarse grained detail is neither desired. A too coarse-grained approach may miss the important characteristics of user behaviour, leading to a non-representative workload, while a too fine grained approach may lead to a workload that is costly to replay and maintain.

To achieve an optimal granularity of user behaviour, prior research often chooses event or action-driven approaches for workload recovery [11], [15]–[18]. However, there exists extensive research on execution log analysis that demonstrates the value of considering contextual information and sequence of actions for various software engineering tasks [25]–[34]. Such extensive usage of contextual information and user action sequences in log analysis motivates our approach to leverage the similar information to recover richer workloads from execution logs for generating load tests.

As an experience report, our focus is primarily on exploring whether research approaches work in practice. Based on our industrial experience, this was not the case. Hence we had to propose two novel approaches. In particular compared to prior research, our work uses more valuable contextual and action sequence information from execution logs to recover workloads. The next section presents our three approaches to cluster user actions, contextual information and user action

TABLE I
OUR RUNNING EXAMPLE OF EXECUTION LOG LINES WITH EXTRACTED USER ACTIONS AND CONTEXT VALUES.

Timestamp	User	Log line	Action	Byte
00:00.00	Alice	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	7204
00:00.92	Dan	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	7216
00:01.52	Bob	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	1249
00:01.54	Alice	DELETE //192.168.165.219:8080/openmrs/ws/rest/v1/person	Delete	0
00:02.04	Alice	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	7227
00:02.26	Bob	DELETE //192.168.165.219:8080/openmrs/ws/rest/v1/person	Delete	0
00:02.41	Bob	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/addPerson	Add	2008
00:02.58	Dan	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/addPerson	Add	8109
00:03.42	Bob	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	1247
00:03.78	Alice	DELETE //192.168.165.219:8080/openmrs/ws/rest/v1/person	Delete	0
00:04.13	Bob	DELETE //192.168.165.219:8080/openmrs/ws/rest/v1/person	Delete	0
00:04.38	Bob	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/addPerson	Add	2010
00:05.69	Dan	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	7213
00:06.06	Alice	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	7231
00:07.31	Dan	DELETE //192.168.165.219:8080/openmrs/ws/rest/v1/person	Delete	0
00:07.41	Dan	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	7221
00:07.81	Alice	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/addPerson	Add	2006
00:09.08	Dan	DELETE //192.168.165.219:8080/openmrs/ws/rest/v1/person	Delete	0
00:10.18	Bob	GET //192.168.165.219:8080/openmrs/ws/rest/v1/person	Search	1251
00:10.32	Alice	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/addPerson	Add	8121
00:10.52	Dan	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/addPerson	Add	2012
00:11.02	Bob	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/editPerson	Edit	868
00:11.47	Dan	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/editPerson	Edit	881
00:12.12	Alice	POST //192.168.165.219:8080/openmrs/ws/rest/v1/person/editPerson	Edit	877

sequences from execution logs for recovering a workload for load testing.

III. OUR APPROACHES OF RECOVERING A WORKLOAD FOR LOAD TESTING

In this section, we present our approaches to automatically recover workloads using system execution logs. An overview of our recovery process is shown in Figure 1. In total, our recovery process consists of six steps: A) extracting user actions, B) enriching user actions with context, C) identifying frequent action sequences, D) grouping similar frequent action sequences, E) grouping users into clusters and F) generating load tests. Our *Action* workloads are generated by steps A, E and F of our recovering process. Our *ActionContext* workloads are generated by steps A, B, E and F. Our *ActionSequence* workloads are generated by all the steps.

A. Extracting user actions

User actions are a typical source of information to recover workload [11]. Therefore, in this step, we extract user actions from execution logs that are generated during the execution of a software system. Execution logs record system events at runtime to enable monitoring, remote issue resolution and system understanding [35]. Each line of an execution log contains a corresponding action and its contextual information (e.g., user names and data sizes). In this step, we first parse the execution logs by identifying the actions and their contextual information. For example, in Table I, we extract four types of actions from the logs that are generated by OpenMRS. We also extract the *Timestamp*, the *User* and the *Byte* values as the contextual values of each action. After extracting user actions, the workload signature of each user can be represented by one point in an n-dimensional space (where n is the total number of extracted actions), i.e., the n-dimension vector for each user records the number of occurrence of each action with each action type mapped to one dimension. Such vectors are directly fed into step E to recover our *Action* workload. Table II shows the vectors of the frequency of user actions from our running example.

TABLE II
FREQUENCY OF ACTIONS FOR USERS IN OUR RUNNING EXAMPLE FOR THE *Action* WORKLOADS.

	User actions			
	Search	Add	Edit	Delete
Alice	3	2	1	2
Bob	3	2	1	2
Dan	3	2	1	2

TABLE III
FREQUENCY OF ENRICHED ACTIONS (WITH CONTEXT) FOR USERS IN OUR RUNNING EXAMPLE FOR THE *ActionContext* WORKLOADS.

	Enriched actions					
	Search1	Search2	Add1	Add2	Edit1	Delete1
Alice	0	3	1	1	1	2
Bob	3	0	2	0	1	2
Dan	0	3	1	1	1	2

B. Enriching user actions with context

Each instance of a user action is associated with a context, which may contain useful information to represent workload. For example, a disk read event is often associated with the size of the read. However, a disk read event with a large size versus one with a small size of data may correspond to different user behaviours. A small disk read may correspond to a user reading a data index while a large disk read may correspond to the actual data reading. Therefore, in this step, we enrich the recovered user actions by considering their context values. In particular, we split each action into multiple ones by categorizing the context values. For example, a disk read action may become two different actions, i.e., a large_read_disk and a small_read_disk. In particular, we use Jenks Natural Breaks [36] on the context values of each action. Jenks Natural Breaks is a one-dimension number clustering algorithm, which minimizes each class's average deviation from the class mean, while maximizing each class's deviation from the means of the other clusters. In our running example, we consider the *Byte* value of each action as its context and for the *Search* action, we split it into two actions: *Search1* (1,247 bytes to 1,251 bytes) and *Search2* (7,204 bytes to 7,231 bytes).

After this step, the workload signature of each user becomes a vector where each dimension is the number of occurrence of each enriched user action. Such vectors are directly fed into step E to recover our *ActionContext* workload. Table III show the vectors of enriched user actions from our running example.

C. Identifying frequent action sequences

Users often perform multiple actions frequently together. In order to capture these actions, we identify action sequences that frequently appear during system execution.

1) Splitting user action sequences: We first group all user actions by each user and sort the actions by their timestamp, in order to generate a sequence of actions for each user. Such sequences are often very long, consisting of thousands of actions; while each user may not perform all the action at once, i.e., a user may perform two series of actions with a long period of idling time in between. Therefore, we wish to split the long user action sequences into smaller ones. One naive approach to split long user action sequences is to consider the

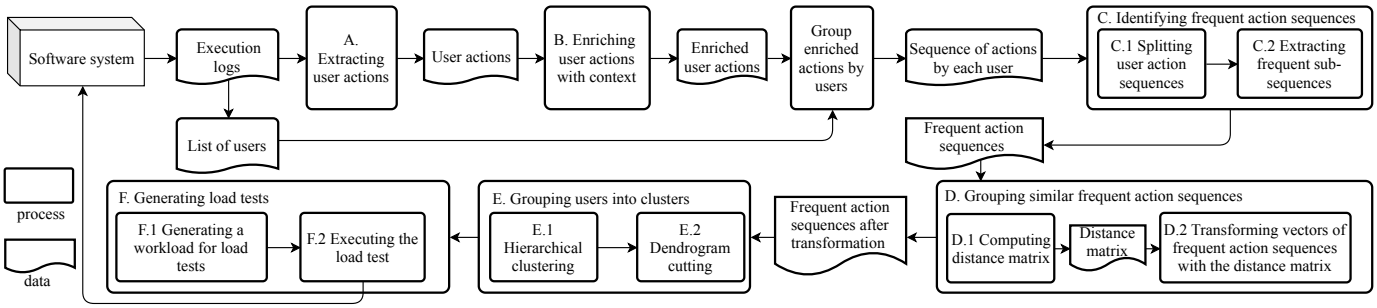

Fig. 1. An overview of our workload recovery process.

time interval between two actions, i.e., if there is a long time interval between two actions, the sequence is split into two actions. However, some actions may actually require a long time to finish, leading to the long wait between two actions. In such cases, the naive approach may incorrectly split the sequences. Therefore, we wish to identify the actions, where the long running time is due to the idling between two septate actions. We leverage a heuristic that considers the relationship between context values and the time interval of each user action as an indicator of the type of wait.

For example, a data read action with a large data size may take longer time than a small size. Based on this intuition, we build a linear regression model using the associated context values of each action as independent variables and its time interval to the next action as the dependent variable. With the linear regression model, if an action has a time interval higher than the predicted value with a residual that is greater than 50%, our process considers that the user has idled after the action. In our running example, we split user Dan's events into three sequences: *Search2→Delete1→Search2→Delete1*, *Search2→Add2* and *Add1→Edit1*.

2) Extracting frequent sub-sequences of actions: We aim to mine frequent sub-sequences in order to represent the series of actions that a user may often perform. An example frequent sub-sequence can be a user first reading small data (to locate the data using an index) before repetitively reading large data (reading the actual data), i.e., *Small_Read→Large_Read→Large_Read→Large_Read*. We apply a suffix array structure and the Longest Common Prefix (LCP) algorithm to mine frequent sub-sequences for each sequence of user actions. Such an approach has been used in prior research to uncover usage patterns from execution logs [37]. We re-implemented the same algorithm as prior research [37]. Due to the limited space, our detailed implementation can be found in our replication package.[1]. In our running example, one of the frequent sub-sequences that we extract is *Search2→Delete1*, which is identified originally from the sequence *Search2→Delete1→Search2→Delete1*.

Since some sub-sequences may be trivial (too short) or do not frequently appear (too rare), we rank the extracted sub-sequences based on the frequency of their occurrence and the frequency of events in the actual sub-sequence, as follows

$$rank = \alpha \times \#occurrence + (1 - \alpha) \times \#events \quad (1)$$

where $\#occurrence$ is the frequency of a sub-sequence's occurrence and $\#events$ is the number of events in the sub-sequence. α is a weight factor for the number of sub-sequence's occurrence. We determine α as 0.5 since we consider the $\#occurrence$ and $\#events$ to be equally important. We use the $rank$ value to keep the top sub-sequences such that the kept sub-sequences cover more than 90% of all actions. We call these kept sub-sequences as frequent action sequences.

After extracting the frequent action sequences, the workload of each user is represented by one point in an n-dimensional space (where n is the total number of identified frequent action sequences), i.e., a vector for each user where each dimension is the frequency of each frequent action sequence. Table IV shows the result of frequent action sequences in our example.

TABLE IV
FREQUENCY OF FREQUENT ACTION SEQUENCES IN OUR RUNNING EXAMPLE FOR THE *ActionSequence* WORKLOADS.

	Frequent action sequences						
	Add1→ Edit1	Add2→ Edit1	Search1→ Delete1→ Add1	Search1→ Edit1	Search2→ Add1	Search2→ Add2	Search2→ Delete1
Alice	0	1	0	0	1	0	2
Bob	0	0	2	1	0	0	0
Dan	1	0	0	0	0	1	2

D. Grouping similar frequent action sequences

The extracted frequent action sequences are not independent from each other. Intuitively, for example, two frequent action sequences *Read1→Read2→Read2→Read1* and *Read1→Read2→Read1* are similar. One user may only have *Read1→Read2→Read2→Read1* and another user may only have *Read1→Read2→Read1*. The two users may be considered completely different if we do not consider the similarities between the two frequent action sequences.

1) Computing distance matrix: For all the frequent action sequences, we calculate the distance between each pair of them using the normalized Levenshtein distance [38]. For example, the normalized Levenshtein distance between *Read1→Read2→Read2→Read1* and *Read1→Read2→Read1* is 0.75.

2) Transforming vectors of frequent action sequences with the distance matrix: In order to address the similarities be-

[1]https://github.com/senseconcordia/ASE2019Data.

TABLE V
RESULT OF FREQUENT ACTIONS SEQUENCES AFTER TRANSFORMATION BASED ON DISTANCE MATRIX FOR *ActionSequence* WORKLOAD.

	Frequent action sequences						
	Add1→ Edit1	Add2→ Edit1	Search1→ Delete1→ Add1	Search1→ Edit1	Search2→ Add1	Search2→ Add2	Search2→ Delete1
Alice	0.5	1	1	0.5	2	1.5	2.5
Bob	0.5	0.5	2.33	1.67	0.67	0	0.67
Dan	1	0.5	0.67	0.5	1.5	2.0	2.5

tween frequent action sequences, we apply a vector transformation based on the distance matrix as follows:

$$vector^u = \left\langle \; s_1^u, \quad \ldots \quad , s_n^u \; \right\rangle \begin{bmatrix} d_1^1, & d_1^2 & \ldots & d_1^n \\ d_2^1, & d_2^2 & \ldots & d_2^n \\ \vdots & & & \\ d_n^1, & d_n^2 & \ldots & d_n^n \end{bmatrix}$$

where $vector^u$ is the final vector for user u, s_n^u is the frequency of frequent action sequence n for user u and d_n^1 is the normalized Levenshtein distance between frequent action sequence 1 and frequent action sequence n. We perform a vector transformation in our running example and the result is shown in Table V.

E. Grouping users into clusters

In order to reach a desirable level of granularity for a workload, in this step, we apply a clustering algorithm to group users into clusters.

1) Hierarchical clustering: We apply a hierarchical clustering to cluster users based on the Pearson distance. We choose hierarchical clustering since it is suitable for data with arbitrary shape and there is no need to determine a specific number of clusters beforehand [39]. In addition, hierarchical clustering performs well in prior research of workload recovery [11], [40]. In our approach, hierarchical clustering first considers each user as an individual cluster. Afterwards, we merge the most neighbouring clusters into a new cluster and recalculate the Pearson distance matrix between each two clusters based on average linkage.

2) Dendrogram cutting: Hierarchical clustering can be visualized using a dendrogram. Such a dendrogram must be cut with a horizontal line at a particular height to create our clusters. In practice, one may choose a desired level of granularity in a recovered workload by cutting the dendrogram at different heights, in order to retrieve a different number of clusters. To avoid any subjective bias, we use the Calinski-Harabasz stopping rule [41] to perform our dendrogram cutting. Prior research notes that the Calinski-Harabasz stopping rule outperforms other rules when clustering workload signatures [42]. The Calinski-Harabasz index measures the dissimilarity of the intra-cluster variance and the similarity of inter-cluster variance. In our running example, Alice, Bob and Dan are all grouped into one cluster for the *Action* workload. Users Alice and Dan are grouped into one cluster while Bob is in another cluster for the *ActionSequence* and *ActionContext* workloads.

F. Generating load tests

In the final step of our process, we generate the load tests as the final outcome of our approach.

1) Generating a workload for load tests: We identify a representative user in each cluster of users to generate a workload for load testing. We apply the Partitioning Around Medoids [43] (PAM) algorithm to identify the representative point of each cluster. PAM is based on the k representative medoids among the instances of the clustering data. PAM is an iterative process of replacing representative instances by other instances until the quality of the resulting clustering cannot be improved. The quality is measured by the medoids with the smallest average dissimilarity to all other points. In our case, we set the k as 1 since we only choose one user to represent each cluster. We then iterate each user inside the cluster to find the best representative user based on PAM.

After obtaining a representative user for each cluster, we obtain a vector $\left\langle \; s_1^u, \quad \ldots \quad , s_n^u \; \right\rangle$ for that user where s_n^u is the number of occurrences of frequent action sequences n from user u, for the *ActionSequence* workload. We use the frequency of occurrences of each frequent action sequences from the representative user to calculate a probability of occurrence of that frequent action sequence. Then, we generate the synthesized workload based on such probability of frequent action sequences. In our running example, the center of the cluster that consists of users Alice and Dan is user Dan. Then to generate a workload for user Alice in the load test, we replace the corresponding actions into *Search2→Delete1* with a probability of 50%, *Search2→Add1* with a probability of 25% and *Add2→Edit1* with a probability of 25%, because each of them has a frequency of two, one and one, respectively (see Table IV).

For the *Action* and the *ActionContext* workload, this step is similar to above but instead a probability of having an action or enriched action is calculated. For the *Action* workload, since all the users are in one cluster, the load test is generated based on the probability of having each action shown in Table II, i.e., the *Search*, *Add*, *Edit* and *Delete* actions have a probability of 37.5%, 25%, 12.5% and 25%, respectively. For the *ActionContext* workload, users Alice and Dan are in one group with exactly the same distribution frequency of actions with context values. Therefore, the generated load test where the *Search2* action has a probability of 37.5%, the *Add1*, *Add2*, and *Edit1* actions each have a probability of 12.5%, and the *Delete1* action has a probability of 25%.

2) Executing load tests: Finally, our approach executes the load tests based on FIO [44] and JMeter [45]. For software systems that cannot be directly driven by FIO [44] and JMeter [45], our approach outputs simulated execution logs. Such systems can generate the load test by directly replaying the workload based on our simulated execution logs line by line.

IV. CASE STUDY SETUP

In this section, we present the setup of our case study.

A. Subject systems

We choose two open-source systems including Apache James and OpenMRS, as well as two industrial systems

TABLE VI
OVERVIEW OF OUR SUBJECT SYSTEMS.

Subjects	Version	SLOC (K)	# Users	# lines of logs (K)
Apache James	2.3.2.1	37.6	2,000	134
OpenMRS	2.0.5	67.3	655	231
Google Borg	May 2011	N/A	4,895	450
SA	2018	N/A	$\gg$5,000	$\gg$1,500

including Google Borg, and one large software system (SA) as our subject systems. Apache James is a Java-based mail server developed by the Apache Foundation. OpenMRS is an open-source health care system to develop to support customized medical records. OpenMRS is a web system developed using the Model-View-Controller (MVC) architecture. Google Borg is a large-scale cluster management system that are used to manage internal workloads and schedule machines, jobs and tasks. SA is an ultra-large-scale cloud computing service application that is deployed to support business worldwide and used by a hundred of millions of users. Due to a Non-Disclosure Agreement (NDA), we cannot reveal additional details about the system. We do note that the SA system is one of the largest in the world in its domain with a long development history. All our subject systems cover different domains and are studied in prior research [46]–[49]. The overview of the four subject systems is shown is Table VI.

B. Data collection

In this subsection, we present how we collect execution logs in order to study the use of our different workload approaches. In particular, for the two open source systems (Apache James and OpenMRS), we deployed the systems in our experimental environment and conducted load tests to exercise the systems. We then collected execution logs that are generated during the load tests. We also collected the CPU usage of both systems by monitoring the particular process of the system with a performance monitoring tool named *Pidstat* [50] for every five seconds. The data from the two industrial systems are from real end users. The production deployment of SA provides the CPU usage of the system, while Google Borg does not provide the CPU usage (hence, we do not evaluate that aspect for this system). We discuss the details of our data collection for each of our subject systems. The details of our data collection can be found in our replication package.

a) Apache James: We use JMeter to create load tests that exercise Apache James. We replicate a similar workload to prior research [46]. In particular, we simulated 2,000 email users who send, receive and read different sizes of emails with and without different sizes of attachment. Users may only read the email header or load the entire email.

We deploy Apache James on a server machine with an Intel Core i7-8700K CPU (3.70GHz), 16 GB memory on a 3TB SATA hard drive. We run JMeter on a machine with Intel Core i5-2400 CPU (3.10GHz), 8 GB memory and 320GB SATA hard drive to generate a two-hours workload.

b) OpenMRS: We used the default OpenMRS demo database in our load tests. The demo database contains data for over 5K patients and 476K observations. OpenMRS contains four typical scenarios: adding, deleting, searching and editing operations. We designed the load tests that are composed of various searches of patients, concepts, encounters, and observations, and addition, deletion and editing of patient information. To simulate a more realistic workload, we added random controllers in JMeter to vary the workload.

We deployed the OpenMRS on two machines, each with Intel Core i5-2400 CPU (3.10GHz), 8 GB memory, 512GB SATA hard drive. One machine is deployed as application server and the other machine as a MySQL database server. OpenMRS provides a web-based interface and RESTFul services. We used the RESTFul API from OpenMRS and ran JMeter on another machine with the same specification to simulate users in the client side with an eight-hours workload.

c) Google Borg: We used a publicly-available open dataset from the Google Borg system [51]. The data is published by Google with a goal of workload related research. Due to the large size of Google Borg data, we only picked the first part of data to analyze, which consists of 83 minutes of data from the entire Google Borg cluster. Due to the inaccessibility of the Google Borg system, we could not run load tests directly on the system. In the data from Google Borg, there exists no information about users. However, the workload is described as *jobs*. Therefore, we considered each job as a user when applying our approach on the Google Borg data.

d) SA: We retrieved the execution logs and the corresponding CPU usage from SA that is deployed in production and is used by real users. The SA is deployed on a cluster with more than a thousand machines. Due to the NDA, we cannot mention the detailed information of the hardware environment and the usage scenarios of SA.

C. Preliminary analysis: clustering tendency

Before we answer the research questions for our case study, we first conduct a preliminary analysis on the clustering tendency of the data from our subject systems. If the users from our subject systems have random behaviour and do not appear to have inherent groups of similar behaviours, the data from our subject systems would be unsuitable for our study.

Therefore, we calculated Hopkins Statistic to assess the cluster tendency of our data. Hopkins Statistic is a statistical hypothesis test that can be used to accept or reject the random position hypothesis [52]. The value of the Hopkins Statistic ranges from 0 to 1. A value of 1 means that the data has a high cluster-tendentious, a value of 0 indicates the data is uniformly distributed (not cluster-tendentious). Similar to previous research [52], we used 0.5 as the threshold to reject the alternative hypothesis. If the value of the Hopkins Statistic is higher than 0.5, we consider that the data has a high cluster-tendentious. We used the function *hopkins* of the *clustertend* package in R to calculate the Hopkins Statistic. We observe that our data has a high cluster-tendentious with Hopkins Statistic values that range between 0.80 to 0.99 with an average of 0.92 across all of our subject systems. Since the Hopkins Statistic values are higher than 0.5, we reject the alternative hypothesis and confirm that our data is suitable for our study.

V. Case study results

In this section, we present our case study results by answering two research questions.

RQ1: How field-representative are our generated workloads?

Motivation. In order to illustrate a practical impact, we wish to first examine whether the generated workloads can lead to similar system behaviour and performance as the original system workload. If the system behaviour and performance that are produced by the generated workloads are drastically different from the original workload, such automatically generated workloads are not field-representative and hence would not be useful in practice.

Approach. We use all three workload approaches (*Action*, *ActionContext* and *ActionSequence*) to generate load tests based on the execution logs of the subject systems. The execution logs from Google Borg do not contain any context values, hence we only use the *Action* and *ActionSequence* approaches to generate load tests for Google Borg. When running the generated load tests, we monitor the behaviour and the performance of the systems. For the behaviour of the system, we measure the throughput of each type of action for every minute during the execution of the load tests. For the system performance, we measure the CPU usage of the particular process of the system. In the load tests, we use a performance monitoring tool named *Pidstat* [50] to collect the physical performance for every five seconds. For our subjects Apache James, OpenMRS and SA, we are able to run the generated load tests directly on the system to measure system behaviour and performance. However, for the subject Google Borg, we cannot directly run the load tests since we do not have access to the system. Therefore, we cannot measure the system performance. Since we can generate simulated execution logs for load tests, we use the simulated execution logs to compute the throughput of each type of user action.

We perform statistical analysis to examine the existence of significant differences between the generated workloads and the original workload, in terms of the throughput of each action and the CPU usage. We use the Mann-Whitney U test [53] to determine if there exists a statistically significant difference (i.e., p-value < 0.05). We choose the Mann-Whitney U test because it does not enforce any assumptions on the distribution of the data. Reporting only the statistical significance may lead to erroneous results (i.e., if the sample size is very large, p-value can be small even if the difference is trivial). Hence, we use Cliff's delta to quantify the effect size [54]. The smaller the effect sizes, the more similar the workload is to the original workload. Since statistical tests do not consider the trend of the actions, we visualize the differences between the number of each type of actions during execution from the load tests and the original workload using cumulative density graphs.

Results. The load tests from our recovered workloads have similar system behaviour to the original workload. The results of throughput of actions between the original workload and the generated load tests are shown in Table VII. In 11 out of 14 user actions in all the subject systems, at least one workload is field-representative (bold in Table VII). Even the most coarse-grained workload *Action* has a similar system behaviour in five out of 14 user actions; while the most fine-grained workload *ActionSequence* has a similar system behaviour in 10 out of 14 user actions.

TABLE VII
Comparing throughput between the original workload and the generated workloads.

OpenMRS						
Action	Add			Delete		
	Throughput (per minute)	Comparing with original p-value	effect size	Throughput (per minute)	Comparing with original p-value	effect size
Original	143	N/A	N/A	91	N/A	N/A
Action	173	≪0.0001	0.52 (large)	**87**	**≪0.0001**	**0.14 (small)**
ActionContext	105	≪0.0001	0.96 (large)	69	≪0.0001	0.83 (large)
ActionSequence	**155**	**≪0.0001**	**0.23 (small)**	83	**≪0.0001**	**0.26 (small)**
Action	Exit			Search		
	Throughput (per second)	Comparing with original p-value	effect size	Throughput (per second)	Comparing with original p-value	effect size
Original	92	N/A	N/A	131	N/A	N/A
Action	**88**	**≪0.0001**	**0.15 (small)**	110	≪0.0001	0.48 (large)
ActionContext	119	≪0.0001	0.87 (large)	165	≪0.0001	0.88 (large)
ActionSequence	71	≪0.0001	0.45 (medium)	149	≪0.0001	0.43 (medium)
Apache James						
Action	Send			Receive		
	Throughput (per minute)	Comparing with original p-value	effect size	Throughput (per minute)	Comparing with original p-value	effect size
Original	339	N/A	N/A	127	N/A	N/A
Action	253	≪0.0001	0.39 (medium)	213	≪0.0001	0.68 (large)
ActionContext	**318**	**≪0.0001**	**0.30 (small)**	149	≪0.0001	0.44 (medium)
ActionSequence	**338**	**0.002**	**0.18 (small)**	**129**	**≪0.0001**	**0.13 (small)**
SA						
Action	Action A			Action B		
	Throughput (per minute)	Comparing with original p-value	effect size	Throughput (per minute)	Comparing with original p-value	effect size
Original	—	N/A	N/A	—	N/A	N/A
Action	—	0.58	0.89 (large)	—	≪0.0001	0.79(large)
ActionContext	—	**0.58**	**0.06 (trivial)**	—	≪0.0001	0.99(large)
ActionSequence	—	**0.89**	**0.01 (trivial)**	—	**0.49**	**0.17(small)**
Google Borg						
Action	Submit			Schedule		
	Throughput (per minute)	Comparing with original p-value	effect size	Throughput (per minute)	Comparing with original p-value	effect size
Original	4,022	N/A	N/A	3,840	N/A	N/A
Action	**3,985**	**0.82**	**0.02 (trivial)**	**3,971**	**0.84**	**0.02 (trivial)**
ActionSequence	**4,192**	**0.51**	**0.06 (trivial)**	**3,849**	**0.98**	**0.003 (trivial)**
Action	Fail			Finish		
	Throughput (per minute)	Comparing with original p-value	effect size	Throughput (per minute)	Comparing with original p-value	effect size
Original	613	N/A	N/A	2,236	N/A	N/A
Action	**647**	**0.006**	**0.25 (small)**	2,679	0.007	0.42 (medium)
ActionSequence	**608**	**0.68**	**0.04 (trivial)**	1,793	0.007	0.45 (medium)
Action	Evict			Kill		
	Throughput (per minute)	Comparing with original p-value	effect size	Throughput (per minute)	Comparing with original p-value	effect size
Original	933	N/A	N/A	217	N/A	N/A
Action	428	0.06	0.44(medium)	151	≪0.0001	0.34 (medium)
ActionSequence	1,258	0.05	0.31(medium)	**162**	**0.37**	**0.08 (trivial)**

Note: Bold font indicates that the load tests from the corresponding approaches are representative to the original workload (p-value <0.05, or effect sizes trivial or small).

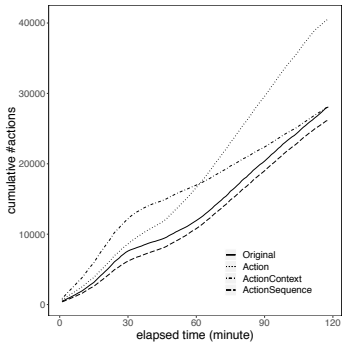

Fig. 2. Cumulative density plot of the number of user actions from the original workload and the generated load tests for the *Receive* action in Apache James.

TABLE VIII
COMPARING CPU USAGE BETWEEN ORIGINAL WORKLOAD AND THE LOAD TESTS GENERATED BY OUR WORKLOAD APPROACHES.

	Comparing with original					
	Action		*ActionContext*		*ActionSequence*	
Subjects	p-value	effect size	p-value	effect size	p-value	effect size
Apache James	$\ll$0.0001	0.63 (large)	$\ll$0.0001	0.98 (large)	$\ll$**0.0001**	**0.18 (small)**
OpenMRS	$\ll$0.0001	0.78 (large)	$\ll$0.0001	0.61 (large)	$\ll$**0.0001**	**0.26 (small)**
SA	$\ll$ 0.0001	0.55 (large)	$\ll$0.0001	0.47 (medium)	$\ll$**0.0001**	**0.29 (small)**

Note: The bold fond indicates the most field-representative workload.

The load tests generated by *ActionSequence* outperform the ones from *Action* and *ActionContext*. We observe that in 10 out of 14 user actions, *ActionSequence* outperforms *Action* and *ActionContext* when comparing the throughput of the user actions. For example, for the *Send* action in Apache James, the load test from *ActionSequence* generates 337 actions per minute which is much closer to the original workload (339 actions per second) than *Action* (253 actions per minute) and *ActionContext* (318 actions per minute). We also use cumulative density graph to evaluate the trend of each action between the original and the load tests that are generated from our different workloads. The x-axis of the graphs is the elapsed time of the load tests and the y-axis of the graph is the total number of actions. Due to space limitation, we only present the cumulative density graph for one action of Apache James (see Figure 2). The detailed cumulative density graph for each user action can be found in our replication package. The graphs show that the trend of the *Receive* action from the *ActionSequence* workload is much closer to original workload than the *Action* and *ActionContext* workloads. In particular, Figure 2 shows that the total of number of user actions from the *Action* workload is far from the original.

The *ActionSequence* workloads produce system performance closer to the original workload than the *Action* and *ActionContext* workloads. Shown in Table VIII, the effect sizes are always trivial or small between the CPU usage during the load tests that are generated from the *ActionSequence* workloads and the original workload. On the other hand, the effect sizes of CPU usage differences are medium to large when comparing the original workload with the *Action* and *ActionContext* workloads.

RQ2: How many clusters of users are captured by each of our recovery workload approaches?

Motivation. Our workloads may contain a large number of clusters of users, leading to a too-fine of a granularity for load tests. If our workloads consist of an overwhelming amount of clusters, they would not be useful for practitioners due to the large overhead of developing the infrastructure to execute such workloads, as well as executing and maintaining them.

Approach. We use all the workloads of each of our approaches to generate load tests for the four subject systems. We compare the number of clusters of users that are recovered by each workload. Afterwards, we manually examine each cluster of users to understand the differences between our different workload recovery approaches.

Results. Our approaches do not generate an overwhelming number of clusters. The numbers of generated user clusters are shown in Table IX. Although including richer user information in our workloads, we do not generate an overwhelming number of clusters. In particular, Google Borg and SA are both large-scale systems with a large amount of end users, while our process only generates a maximum of 25 and 13 clusters for the *ActionSequence* approach for Google Borg and SA, respectively. However, the granularity of the *Action* workloads is very coarse. In particular, for the three subject systems, i.e., Apache Jame, OpenMRS and SA, the *Action* workload only consists of two to three clusters of users. Such a small number of clusters helps explain the results from RQ1 where the *Action* workload is not able to generate field-representative load tests. On the other hand, the most field-representative workload from RQ1, i.e., the *ActionSequence* workloads, consists of only three to six more clusters than the *ActionContext* workloads. Such a small number of clusters of users make it possible to manually examine each cluster to qualitatively understand the field workload.

TABLE IX
NUMBER OF USER CLUSTERS FOR EACH OF OUR WORKLOAD APPROACHES.

Subjects	*Action*	*ActionContext*	*ActionSequence*
Apache James	2	35	39
OpenMRS	3	2	8
Google Borg	20	20	25
SA	2	10	13

We manually examine each cluster of users and aim to understand the difference between the recovered clusters for the *ActionSequence* and *ActionContext* workload recovery approaches. We identify three typical scenarios that cause the differences. 1) Orders of actions. Some users have the same distribution of actions but they are ordered differently. Such differences are not captured by *ActionContext* workloads. 2) Similar sequences of actions but different distribution. User may have the same action sequences but the general distribution of each sequence of actions is different. *ActionContext* workloads would consider the users into different clusters while *ActionSequence* workloads group the users together. 3) Varying frequency of actions. Some users have the same distribution of action sequences although their frequencies are varying differently. *ActionContext* workloads would consider the users in the different clusters while *ActionSequence* workloads consider them the same. Such scenarios show that the *ActionSequence* considers a different levels of granularities of user behaviours which may explain the differences in clusters.

VI. DISCUSSION

In this section, we discuss the use of our three workload recovery approaches to detect unseen workloads and a sensitivity analysis.

A. Detecting unseen workload

One of the challenges of designing load tests is keeping the load tests field-representative as the user workloads evolve [55], [56]. When there exist users with unseen workload, practitioners should be informed in order to act accordingly. For example, if the unseen workload is due to

new user behaviours, one may wish to update the load tests. On one hand, if the recovered workloads from our approaches are too fine-grained, our approaches would report false-positively unseen workloads, leading to additional wasted costs to practitioners. On the other hand, if our recovered workloads are too coarse-grained, we may miss the reporting of unseen workloads, leading to load tests that are not field-representative. Therefore, we study the use of our workload recovery process to detect unseen workloads by injecting users with unseen workloads into our existing data. We injected four types of unseen workloads, that are typically used in prior research [56], [57].

- *Extra actions.* We randomly pick one action that is not the most frequent action. Then we replace all occurrences of the picked action by the most frequent action.
- *Removing actions.* We randomly pick an action type and remove all occurrences of the action from a user.
- *Double context values.* For every action with a context value, we change the context value by doubling it.
- *Reordering actions.* For all the actions that are performed by a user, we randomly reorder the actions.

For every type of unseen workload, we randomly select one user and alter the data from the user to inject the unseen workload. We apply each of our approaches to recover workloads from the data with only one user injected with an unseen workload. In particular, if one user is located in one cluster without any other users, we consider that the user has an unseen workload (the user is not similar to any other one). If the user is indeed injected with an unseen workload, we consider it as a true-positive detection. Any normal user located in a one-user cluster will be considered as a false-positive detection. We repeat this process 10 times, with every time injecting a random user with an unseen workload, for every type of unseen workload. In total, for our four subject systems, we have 160 sets of data, each of them having one user injected with an unseen workload. We generate 160 workloads to detect those users. We define precision as the number of the true-positive detection divided by the total number of users that are in a one-user cluster; recall as the number of the true-positive detection divided by the total number of users with injected unseen workloads.

***ActionContext* and *ActionSequence* workloads can accurately detect the injected users with an unseen workload.** The results of detecting injected users with an unseen workload is shown in Table X. The results show that *ActionContext* workloads have a precision between 0.75 to 1 with an average of 0.86 and a recall between 0.2 to 0.9 with an average of 0.58. The *ActionSequence* workloads achieve a precision between 0.75 to 1 with an average of 0.87 and a recall between 0.3 to 1 with an average of 0.79. However, the *Action* workloads have both low average precision (0.47) and recall (0.29). The precision and recall of the *ActionSequence* workloads are generally consistently high across all subject systems for all types of injected unseen workloads. The only exception is Apache Jame with a lower recall in detecting *Extra actions* and *Removing actions* workloads. We find that the Apache James mail server has a small number of action types (cf. Section IV), while adding extra actions and removing actions may still generate a user with a high similarity to a previously recovered cluster of users.

TABLE X
RESULTS OF PRECISION AND RECALL IN DETECTING INJECTED UNSEEN WORKLOADS.

Apache James						
	Action		*ActionContext*		*ActionSequence*	
Unseen workload type	precision	recall	precision	recall	precision	recall
Extra actions	**1.00**	0.30	0.80	**0.40**	0.80	**0.40**
Removing actions	**1.00**	**0.40**	0.75	0.30	0.75	0.30
Double context values	0	0	**0.90**	**0.90**	**0.90**	**0.90**
Reordering actions	0	0	0.75	0.30	**0.89**	**0.80**
Average	0.50	0.18	0.80	0.48	**0.84**	**0.60**

OpenMRS						
	Action		*ActionContext*		*ActionSequence*	
Unseen workload type	precision	recall	precision	recall	precision	recall
Extra actions	0.86	0.6	0.86	0.60	**0.89**	**0.80**
Removing actions	**0.88**	0.70	0.73	0.80	0.69	**0.90**
Double context values	0	0	**0.75**	**0.60**	**0.75**	**0.60**
Reordering actions	0	0	**1.00**	0.30	0.91	**1.00**
Average	0.44	0.33	**0.84**	0.58	0.81	**0.83**

Google Borg				
	Action		*ActionSequence*	
Unseen workload type	precision	recall	precision	recall
Extra actions	**0.90**	**0.90**	**0.90**	**0.90**
Removing actions	0.83	0.50	**0.89**	**0.80**
Double context values	**1.00**	0.10	0.90	**0.90**
Reordering actions	**0.91**	0.50	0.90	**0.87**
Average	**0.91**	0.50	0.90	**0.87**

SA						
	Action		*ActionContext*		*ActionSequence*	
Unseen workload type	precision	recall	precision	recall	precision	recall
Extra actions	**1.00**	0.80	0.89	0.80	0.90	**0.90**
Removing actions	0.88	0.7	**0.89**	**0.80**	**0.89**	**0.80**
Double context values	0	0	**1.00**	**0.90**	**1.00**	**0.90**
Reordering actions	0	0	**1.00**	0.20	0.90	**0.90**
Average	0.47	0.38	**0.95**	0.68	0.92	**0.88**

	Action		*ActionContext*		*ActionSequence*	
	precision	recall	precision	recall	precision	recall
All average	0.47	0.29	0.86	0.58	**0.87**	**0.79**

Note: Values in bold font indicate the best performing approach for each setting.

***ActionSequence* workloads have a similar precision but much higher recall than *ActionContext* workloads.** Table X shows that on average *ActionSequence* achieves a similar precision (0.87) as *ActionContext* (0.86) while the recall of *ActionSequence* (0.79) is much higher than that of *ActionContext* (0.56). *ActionSequence* considers finer-grained information than *ActionContext*, hence intuitively, providing a higher ability to uncover more unseen workloads. In particular, in all the cases, *ActionSequence* has a higher or similar recall as *ActionContext*. On the other hand, even though in some cases when *ActionContext* has a higher precision, the difference is rather small.

B. Sensitivity analysis

Our workload recovery process leverages several techniques, such as hierarchical clustering, which may be replaced by other similar techniques. Our process also leverages threshold values. For example, the residual value for the linear regression prediction that is used to split user action sequences is set to 0.5 in our process. The α value to rank frequent action sequences is also set to 0.5. In order to better understand the

TABLE XI
COMPARING THE RESULTS OF CHOOSING DIFFERENT THRESHOLD VALUES AND CLUSTERING ALGORITHM FOR APACHE JAMES.

Send action										
Changed	residual=0.25		residual=0.75		α=0.25		α=0.75		with Mean shift	
Default	p-value	effect size	p-value	effect size	p-value	effect size	p-value	effect size	p-value	effect size
residual=0.5, α=0.5, hierarchical clustering	0.9	0.01 (trivial)	0.91	0.01 (trivial)	0.001	0.19 (small)	0.99	0.0001 (trivial)	0.0001	0.22 (small)
Receive action										
Changed	residual=0.25		residual=0.75		α=0.25		α=0.75		with Mean shift	
Default	p-value	effect size	p-value	effect size	p-value	effect size	p-value	effect size	p-value	effect size
residual=0.5, α=0.5, hierarchical clustering	0.9	0.01 (trivial)	0.9	0.01 (trivial)	0.005	0.15 (small)	0.99	0.0001 (trivial)	0.25	0.07 (trivial)

sensitivity of our workloads to these thresholds, we individually increased each threshold value to 0.75 and decreased the threshold value to 0.25. We also change the clustering algorithm to Mean shift [58]. We choose Mean shift, since similar to hierarchical clustering, Mean shift does not require us to pre-specify the number of clusters. We re-generated the *ActionSequence* workload and calculated the throughput of each action for each subject system. We used the *ActionSequence* workload in this analysis because *ActionSequence* is the best performing workload shown in our evaluation results (c.f. Section V) and the *ActionSequence* covers all the steps of all our three workload recovery approaches.

We compared the throughput of each action with the default threshold and the clustering algorithm using the Mann-Whitney U test and Cliff's delta. We observed that our approaches are insensitive to both the residual and α thresholds. In addition, the effect size between the hierarchical clustering and Mean shift is trivial or small. Table XI only shows the results of such a comparison for Apache James which is the most peculiar workload in our subject systems. Other comparison results are in our replication package.

VII. CHALLENGES AND LESSONS LEARNED FROM THE INDUSTRIAL EVALUATION OF OUR APPROACHES.

In this section we discuss the learned lessons and faced challenges during the implementation and evaluation of our approaches in industry. In particular, the first author of this paper was embedded on site with the industrial team for over half a year to enable a faster feedback loop from practitioner – ensure the smooth adoption of our approaches in a large-scale complex industrial setting. Our documented challenges and lessons can assist researchers and practitioners who would like to integrate their research in complex industrial settings.

A. Domain knowledge is crucial for the successful transfer of research to practice.

Our approaches depend on the availability and quality of the important knowledge that resides in system execution logs. Due to the large scale and complexity of the logs in System A, we often faced the challenge that we may not fully understand the information that is communicated in the logs, making it challenging for us to determine the important contextual information to include and analyze by our approaches.

How to address: At a first attempt, we naïvely included all log information for our analysis. We ended up observing that such an attempt introduced noise which negatively impacted our results. Hence, the first author flew down to spend six months on site at Alibaba, where he held several in-person scrum meetings with developers and operators of System A, in order to better understand the information that is communicated in System As logs. These meetings helped the academic team and the industrial team get a better understanding of the problem at hand as well as the strengths and limitations of the research solutions. Being on site helped the academic team create a strong relation with the industrial team as well. Such a relation enabled a much faster and more open feedback loop.

Lessons learned: Good domain knowledge is crucial in log analysis and workload recovery. Blindly applying log analysis techniques on large-scale complex logs may not achieve the expected goal. One should work closely with practitioners to leverage their valuable knowledge about their logs.

B. Team support is crucial for the successful transfer of research to practice.

1) Customization of tooling: We started our research working on open-source systems. Such open-source systems commonly use standard load driver tools such as JMeter. However, industrial systems are commonly tested using various in-house custom tools. Such tools hinder us from demonstrating and evaluating our work. It is often costly, time-consuming and sometimes impossible to design and implement a specific load replay tool for each system.

How to address: Working closely with the practitioners of Alibaba, we gained a deeper understanding of their in-house load replay tools. We observed that many of them support the direct reading of logs and the replaying of the exact workload based on such logs. Therefore, we provided an option in our toolset for the direct driving of a load test using widely used tools (like JMeter), or the transformation of our generated load test into logs, which can be fed to the customized load replay tools from Alibaba.

Lessons learned: There exists a strong need for future research in load replay using more flexible frameworks in order to avoid practitioners having to implement customized toolsets.

2) Addition of needed log lines when not all information is available: Not all the needed information was available in the logs when we started our research on Alibabas system. In particular, the log data was not perfectly designed for our approach and important information was missing.

How to address: Working closely with the practitioners, we requested the addition of new logging probes. In order to demonstrate the values of our requests, we conducted several additional analyses.

Lessons learned: The technical support and quick turn-around from the industrial team were extremely crucial in addressing this challenge. However, even with all the logging probes in place, we had to wait till the builds with such probes were deployed long enough in the field for us to have sufficient data for our approaches.

3) Setting up a realistic industrial environment: The context of the load testing environment has an important influence on the performed load tests. For example, there should be a large amount of realistic data in a database before one can

test a database-driven application. Using open source systems, it is relatively easy to create a load testing environment that is similar to widely adopted benchmarks. However, setting up such an environment in an ultra-large-scale industrial environment is extremely challenging.

How to address: To make our automatically generated load tests run successfully, we had the luxury of being strongly supported by the infrastructure team of Alibaba. In particular, we were provided with a testing environment that is a replica of the field environment from which we collected the analyzed logs. However, such a solution is not optimal and may not be cost-effective, especially for practitioners that do not have direct access to infrastructures that are similar to their deployed systems.

Lessons learned: Preparing the load testing environment is an important, challenging and yet open problem for load testing practices and research. Technical and infrastructural support is crucial in overcoming this challenge.

C. Coping with the large scale industrial data

Our experiments on open-source systems are conducted with a limited scale of data. When adopting our approach using the industry scale data of Alibaba, our approach did not scale well. The statistical analyses and clustering techniques often suffered from poor scalability.

How to address: In order to ease the adoption of our approach on industry scale data, we optimized our solution by learning some threshold values by pre-processing the logs and caching the thresholds across runs. For example, learning the best threshold values to categorize contextual values is time consuming. We can save the learned thresholds and use them directly to generate the categorized contextual values in logs since such thresholds rarely change within a specific context.

Lessons learned: One should not assume that a successful research tool can directly scale to industrial data. Optimization for the particular industrial setting is important.

VIII. Threats to validity

External validity. Our evaluation is conducted on data from two open source and two industrial systems. Although our subject systems cover different domains and sizes, our evaluation results may still not generalize to other systems. Given the automated nature of our process, others can study its effectiveness on their own data sets using our replication script.

Internal validity. Our approaches depend on the availability and quality of system execution logs. If a system records limited information about user actions in the execution logs, our approaches may not perform as expected. The evaluation of approaches uses the system CPU usage that is recorded by *Pidstat*. The quality and the frequency of recorded CPU usage can impact the internal validity of our study. Currently, our approach only categorizes numerical contextual values due to the characteristics of the logs in our subject systems. Future work can complement our approach by consider categorizing string literals. Our approach depends on various statistical analyses. Therefore, for small systems with a small amount of data, our approach may not perform well due to the nature of statistical analyses.

Construct validity. There exists other aspect of system bahaviour and performance. We focus on the throughput and CPU usage due to special need of SA from our industrial collaborator. Future study may investigate the impact on other system aspects to complement our findings. Due to inaccessibility of the subject system, the workloads on Google Borg is based on simulated execution logs, instead of actually running the load tests on Google Borg. Therefore, the evaluation results may be different if we were able to run the load tests on the actual Google Borg cluster. In the evaluation of our approaches to detect users with unseen workloads, we only injected four types of unseen workloads. Similar evaluation approaches based on mutation techniques have been often used in prior research [59], [60]. However, these unseen workloads may not be the same as in real life. In addition, there may exist other ways to inject unseen workload to complement our results.

IX. Conclusions

Workload recovery from end users is an essential task for the load testing of large-scale systems. In this paper, we conduct a study on recovering workloads at different levels of granularity of user behaviours to automatically generate load tests. We design three approaches that are based on user actions, user actions with contextual information and user action sequences with contextual information, to generate load tests on two open source systems and two industrial systems. We find that our richest approach which uses user action sequences with contextual information outperforms the other two approaches. In most cases, the throughput and CPU usage from the load tests that are generated from the user action sequence based workload outperform the other two, and are statistically insignificant relative to the original workload or with small or trivial effect sizes. Such a field-representative workload is generated only using a small number of clusters of users. In addition, we find that the recovered workloads from our approaches can also be used to detect injected users with unseen workloads with a high precision and recall.

Our paper has the following contributions:

- To the best of our knowledge, our approach is the first large-scale study on the use of different granularity of recovered user details for load testing.
- To the best of our knowledge, our approaches are the first ones in the field to leverage user contextual information and frequent action sequences in workload recovery.
- Our approaches have been adopted in practice to assist in the testing and operation of an ultra-large-scale industrial software system.

Acknowledgement

We would like to thank Alibaba for providing access to their system used in our study. The findings and opinions expressed in this paper are those of the authors and do not necessarily represent or reflect those of Alibaba and/or its subsidiaries and affiliates. Moreover, our results do not reflect the quality of Alibaba's products.

REFERENCES

[1] S. Fiegerman, "Netflix adds 9 million paying subscribers, but stock falls," https://www.cnn.com/2019/01/17/media/netflix-earnings-q4/index.html, January 2019, (Accessed on 03/29/2019).

[2] F. Company, "How one second could cost amazon 1.6 billion in sales," Jul 2012. [Online]. Available: https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales

[3] "Geeking with greg: Marissa mayer at web 2.0," http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html, (Accessed on 04/01/2019).

[4] E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," *IEEE transactions on software engineering*, vol. 26, no. 12, pp. 1147–1156, 2000.

[5] S. Elnaffar and P. Martin, "Characterizing computer systems workloads," *Submitted to ACM Computing Surveys Journal*, 2002.

[6] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Comput. Surv.*, vol. 48, no. 3, pp. 48:1–48:43, 2016.

[7] M. Andreolini, M. Colajanni, and P. Valente, "Design and testing of scalable web-based systems with performance constraints," in *FIRB-Perf Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems (FIRB-Perf 2005), 19 September 2005, Torino, Italy*, 2005, pp. 15–25.

[8] D. Krishnamurthy, J. A. Rolia, and S. Majumdar, "A synthetic workload generation technique for stress testing session-based systems," *IEEE Trans. Software Eng.*, vol. 32, no. 11, pp. 868–882, 2006.

[9] N. Snellman, A. Ashraf, and I. Porres, "Towards automatic performance and scalability testing of rich internet applications in the cloud," 2011, pp. 161–169.

[10] J. A. Meira, E. C. de Almeida, G. Sunyé, Y. L. Traon, and P. Valduriez, "Stress testing of transactional database systems," *JIDM*, vol. 4, no. 3, pp. 279–294, 2013.

[11] M. D. Syer, W. Shang, Z. M. Jiang, and A. E. Hassan, "Continuous validation of performance test workloads," *Automated Software Engineering*, vol. 24, no. 1, pp. 189–231, 2017.

[12] "The workload for the specweb96 benchmark," September 2003. [Online]. Available: https://www.spec.org/web96/workload.html

[13] W. Shang, A. E. Hassan, M. Nasser, and P. Flora, "Automated detection of performance regressions using regression models on clustered performance counters," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15. New York, NY, USA: ACM, 2015, pp. 15–26. [Online]. Available: http://doi.acm.org/10.1145/2668930.2688052

[14] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, ser. SOSP '05. New York, NY, USA: ACM, 2005, pp. 105–118.

[15] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, and H. Krcmar, "WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction - a model-driven approach for session-based application systems," *Software and System Modeling*, vol. 17, no. 2, pp. 443–477, 2018.

[16] J. Summers, T. Brecht, D. Eager, and A. Gutarin, "Characterizing the workload of a netflix streaming video server," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1–12.

[17] H. Xi, J. Zhan, Z. Jia, X. Hong, L. Wang, L. Zhang, N. Sun, and G. Lu, "Characterization of real workloads of web search engines," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2011, pp. 15–25.

[18] A. E. Hassan, D. J. Martin, P. Flora, P. Mansfield, and D. Dietz, "An industrial case study of customizing operational profiles using log compression," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 713–723. [Online]. Available: http://doi.acm.org/10.1145/1368088.1379445

[19] T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Finding and evaluating the performance impact of redundant data access for applications that are developed using object-relational mapping frameworks," *IEEE Trans. Softw. Eng.*, vol. 42, no. 12, pp. 1148–1161, Dec. 2016. [Online]. Available: https://doi.org/10.1109/TSE.2016.2553039

[20] A. Haghdoost, W. He, J. Fredin, and D. H. C. Du, "On the accuracy and scalability of intensive I/O workload replay," in *15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, February 27 - March 2, 2017*, 2017, pp. 315–328.

[21] N. J. Yadwadkar, C. Bhattacharyya, K. Gopinath, T. Niranjan, and S. Susarla, "Discovery of application workloads from network file traces," in *8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 23-26, 2010*, 2010, pp. 183–196.

[22] A. Busch, Q. Noorshams, S. Kounev, A. Koziolek, R. H. Reussner, and E. Amrehn, "Automated workload characterization for I/O performance analysis in virtualized environments," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015*, 2015, pp. 265–276.

[23] B. Seo, S. Kang, J. Choi, J. Cha, Y. Won, and S. Yoon, "IO workload characterization revisited: A data-mining approach," *IEEE Trans. Computers*, vol. 63, no. 12, pp. 3026–3038, 2014.

[24] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, 2017, pp. 153–167.

[25] T. Barik, R. DeLine, S. Drucker, and D. Fisher, "The bones of the system: A case study of logging and telemetry at microsoft," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, May 2016, pp. 92–101.

[26] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM*, vol. 55, no. 2, pp. 55–61, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2076450.2076466

[27] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automatic identification of load testing problems," in *24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28 - October 4, 2008, Beijing, China*, 2008, pp. 307–316.

[28] ——, "Automated performance analysis of load tests," in *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*, 2009, pp. 125–134.

[29] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu, "Logmaster: Mining event correlations in logs of large-scale cluster systems," in *Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems*, ser. SRDS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 71–80. [Online]. Available: http://dx.doi.org/10.1109/SRDS.2012.40

[30] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 267–277. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025151

[31] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with csight," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 468–479. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568246

[32] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 402–411. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486842

[33] 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*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 102–111. [Online]. Available: http://doi.acm.org/10.1145/2889160.2889232

[34] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: ACM, 2018, pp. 60–70. [Online]. Available: http://doi.acm.org/10.1145/3236024.3236083

[35] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 249–267, 2008.

[36] G. F. Jenks, "The data model concept in statistical mapping," *International yearbook of cartography*, vol. 7, pp. 186–190, 1967.

[37] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *ISSRE 2009, 20th International Symposium on Software Reliability Engineering, Mysuru, Karnataka, India, 16-19 November 2009*, 2009, pp. 41–50.

[38] M. Mednis and M. Aurich, "Application of string similarity ratio and edit distance in automatic metabolite reconciliation comparing reconstructions and models," *Biosystems and Information technology*, vol. 1, pp. 14–18, 01 2012.

[39] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[40] W. Shang, A. E. Hassan, M. Nasser, and P. Flora, "Automated detection of performance regressions using regression models on clustered performance counters," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, 2015, pp. 15–26.

[41] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[42] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora, "Continuous validation of load test suites," in *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*. ACM, 2014, pp. 259–270.

[43] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[44] Axboe, "axboe/fio," Mar 2019. [Online]. Available: https://github.com/axboe/fio

[45] "Apache jmeter - apache jmete," https://jmeter.apache.org/, (Accessed on 03/29/2019).

[46] R. Gao, Z. M. Jiang, C. Barna, and M. Litoiu, "A framework to evaluate the effectiveness of different load testing analysis techniques," in *2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, Chicago, IL, USA, April 11-15, 2016*, 2016, pp. 22–32.

[47] T.-H. Chen, W. Shang, A. E. Hassan, M. Nasser, and P. Flora, "Cacheoptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 666–677.

[48] T. M. Ahmed, C.-P. Bezemer, T.-H. Chen, A. E. Hassan, and W. Shang, "Studying the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: an experience report," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 1–12.

[49] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[50] "pidstat(1): Report statistics for tasks - linux man page," https://linux.die.net/man/1/pidstat, (Accessed on 04/06/2019).

[51] "Google ai blog: More google cluster data," https://ai.googleblog.com/2011/11/more-google-cluster-data.html, (Accessed on 04/04/2019).

[52] A. Banerjee and R. N. Dave, "Validating clusters using the hopkins statistic," in *2004 IEEE International Conference on Fuzzy Systems (IEEE Cat. No. 04CH37542)*, vol. 1. IEEE, 2004, pp. 149–153.

[53] N. Nachar *et al.*, "The mann-whitney u: A test for assessing whether two independent samples come from the same distribution," *Tutorials in Quantitative Methods for Psychology*, vol. 4, no. 1, pp. 13–20, 2008.

[54] L. A. Becker, "Effect size (es)," *Accessed on October*, vol. 12, no. 2006, pp. 155–159, 2000.

[55] T. Chen, M. D. Syer, W. Shang, Z. M. Jiang, A. E. Hassan, M. N. Nasser, and P. Flora, "Analytics-driven load testing: An industrial experience report on load testing of large-scale systems," in *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017*, 2017, pp. 243–252.

[56] C. A. Cunha and L. M. e Silva, "Separating performance anomalies from workload-explained failures in streaming servers," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, 2012, pp. 292–299.

[57] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008, pp. 452–461.

[58] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002. [Online]. Available: http://dx.doi.org/10.1109/34.1000236

[59] J. Svajlenko, C. K. Roy, and J. R. Cordy, "A mutation analysis based benchmarking framework for clone detectors," in *2013 7th International Workshop on Software Clones (IWSC)*. IEEE, 2013, pp. 8–9.

[60] J. Svajlenko and C. K. Roy, "Evaluating clone detection tools with bigclonebench," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 131–140.