Autom Softw Eng (2017) 24:189–231
DOI 10.1007/s10515-016-0196-8

Continuous validation of performance test workloads

Mark D. Syer[1] · Weiyi Shang[1] ·
Zhen Ming Jiang[2] · Ahmed E. Hassan[1]

Received: 30 June 2014 / Accepted: 14 March 2016 / Published online: 1 April 2016

Abstract The rise of large-scale software systems poses many new challenges for the software performance engineering field. Failures in these systems are often associated with performance issues, rather than with feature bugs. Therefore, performance testing has become essential to ensuring the problem-free operation of these systems. However, the performance testing process is faced with a major challenge: evolving field workloads, in terms of evolving feature sets and usage patterns, often lead to "outdated" tests that are not reflective of the field. Hence performance analysts must continually validate whether their tests are still reflective of the field. Such validation may be performed by comparing execution logs from the test and the field. However, the size and unstructured nature of execution logs makes such a comparison unfeasible without automated support. In this paper, we propose an automated approach to validate whether a performance test resembles the field workload and, if not, determines how they differ. Performance analysts can then update their tests to eliminate such differences, hence creating more realistic tests. We perform six case studies on two large systems: one open-source system and one enterprise system. Our approach identifies

✉ Mark D. Syer
mdsyer@cs.queensu.ca

Weiyi Shang
swy@cs.queensu.ca

Zhen Ming Jiang
zmjiang@cse.yorku.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

1 Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Kingston, Canada

2 Department of Electrical Engineering & Computer Science, York University, Toronto, Canada

differences between performance tests and the field with a precision of 92 % compared to only 61 % for the state-of-the-practice and 19 % for a conventional statistical comparison.

Keywords Performance testing · Continuous testing · Workload characterization · Workload comparison · Execution logs

1 Introduction

The rise of large-scale software systems (e.g., Amazon.com and Google's GMail) poses new challenges for the software performance engineering field (Software Engineering Institute 2006). These systems are deployed across thousands of machines, require near-perfect up-time and support millions of concurrent connections and operations. Failures in such systems are more often associated with performance issues, rather than with feature bugs (Weyuker and Vokolos 2000; Dean and Barroso 2013). These performance issues have led to several high-profile failures, including (1) the inability to scale during the launch of Apple's MobileMe (Cheng 2008), the release of Firefox 3.0 (SiliconBeat 2008) and the United States government's roll-out of healthcare.gov (Bataille 2013), (2) costly concurrency defects during the NASDAQ's initial public offering of shares in Facebook (Benoit 2013) and (3) costly memory leaks in Amazon Web Services (Williams 2012). Such failures have significant financial and reputational repercussions (Harris 2011; Coleman 2011; Ausick 2012; Howell and Dinan 2014).

Performance testing has become essential in ensuring the problem-free operation of such systems. Performance tests are usually derived from the field (i.e., alpha or beta testing data or actual production data). The goal of such tests is to examine how the system behaves under realistic workloads to ensure that the system performs well in the field. However, ensuring that tests are "realistic" (i.e., that they accurately reflect the current field workloads) is a major challenge. Field workloads are based on the behaviour of thousands or millions of users interacting with the system. These workloads continuously evolve as the user base changes, as features are activated or disabled and as user feature preferences change. Such evolving field workloads often lead to tests that are not reflective of the field (Bertolotti and Calzarossa 2001; Voas 2000). Yet the system's behaviour depends significantly on the field workload (Zhang et al. 2013; Dean and Barroso 2013).

Performance analysts monitor the impact of field workloads on the system's performance using performance counters (e.g., response time and memory usage) and reliability counters (e.g., mean time-to-failure). Performance analysts must determine the cause of any deviation in the counter values from the specified or expected range (e.g., response time exceeds the maximum response time permitted by the service level agreements or memory usage exceeds the average historical memory usage). These deviations may be caused by changes to the field workloads (Zhang et al. 2013; Dean and Barroso 2013). Such changes are common and may require performance analysts to update their tests (Bertolotti and Calzarossa 2001; Voas 2000). This has led to the emergence of "continuous testing," where tests are continuously updated and re-run even after the system's deployment.

A major challenge in the continuous testing process is to ensure performance tests accurately reflect the current field workloads. However, documentation describing the expected system behaviour is rarely up-to-date (Parnas 1994). Fortunately, execution logs, which record notable events at runtime, are readily available in most large-scale systems to support remote issue resolution and legal compliance. Further, these logs contain developer and operator knowledge (i.e., they are manually inserted by developers) whereas instrumentation tends to view the system as a black-box (Shang et al. 2011, 2015). Hence, execution logs are the best data available to describe and monitor the behaviour of the system under a realistic workload. Therefore, we propose an automated approach to validate performance tests by comparing system behaviour across tests and the field. We derive workload signatures from execution logs, then use statistical techniques to identify differences between the workload signatures of the performance test and the field.

Such differences can be broadly classified as feature differences (i.e., differences in the exercised features), intensity differences (i.e., differences in how often each feature is exercised) and issue differences (i.e., new errors appearing in the field). These identified differences can help performance analysts improve their tests in the following two ways. First, performance analysts can tune their performance tests to more accurately represent current field workloads. For example, the performance test workloads can be updated to eliminate differences in how often features are exercised (i.e., to eliminate intensity differences). Second, new field errors, which are not covered in existing testing, can be identified based on the differences. For example, a machine failure in a distributed system may raise new errors that are often not tested.

This paper makes three contributions:

1. We develop an automated approach to validate the representativeness of a performance test by comparing the system behaviour between tests and the field.
2. Our approach identifies important execution events that best explain the differences between the system's behaviour during a performance test and in the field.
3. Through six case studies on two large systems, one open-source system and one enterprise system, we show that our approach is scalable and can help performance analysts validate their tests.

This paper extends our previous research comparing the behaviour of a system's users, in terms of feature usage expressed by the execution events, between a performance test and the field (Syer et al. 2014). We have improved our approach with statistical tests to ensure that we only report the execution events that best explain the differences between a performance test and the field. We have also extended our approach to compare the aggregate user behaviour in addition to the individual user behaviour. Finally, we have significantly improved the empirical evaluation of our approach and the scope of our case studies.

1.1 Organization of the paper

This paper is organized as follows: Sect. 2 provides a motivational example of how our approach may be used in practice. Section 3 describes our approach in detail. Section 4 presents our case studies. Section 5 discusses the results of our case studies and some of the design decisions for our approach. Section 6 outlines the threats to validity and Sect. 7 presents related work. Finally, Sect. 8 concludes the paper and presents our future work.

2 Motivational example

Jack, a performance analyst, is responsible for continuously performance testing a large-scale telecommunications system. Given the continuously evolving field workloads, Jack often needs to update his performance tests to ensure that the test workloads reflect, as much as possible, the field workloads. Jack monitors the field workloads using performance counters (e.g., response time and memory usage). When one or more of these counters deviates from the specified or expected range (e.g., response time exceeds the maximum response time specified in the service level agreements or memory usage exceeds the average historical memory usage), Jack must investigate the cause of the deviation. He may then need to update his tests.

Jack monitors the system's performance in the field and discovers that the system's memory usage exceeds the average historical memory usage. Pressured by time (given the continuously evolving nature of field workloads) and management (who are keen to boast a high quality system), Jack needs to quickly update his performance tests to replicate this issue in his test environment. Jack can then determine why the system is using more memory than expected. Although the performance counters have indicated that the field workloads have changed (leading to increased memory usage), the only artifacts that Jack can use to understand *how* the field workloads have changed, and hence how his tests should be updated, are execution logs. These logs describe the system's behaviour, in terms of important execution events (e.g., starting, queueing or completing a job), during the test and in the field.

Jack tries to compare the execution logs from the field and the test by looking at how often important events (e.g., receiving a service request) occur in the field compared to his test. However, terabytes of execution logs are collected and some events occur millions of times. Further, Jack's approach of simply comparing how often each event occurs does not provide the detail he needs to fully understand the differences between the field and test workloads. For example, simply comparing how often each event occurs ignores the use case that generated the events (i.e., the context).

To overcome these challenges, Jack needs an automated, scalable approach to determine whether his tests are reflective of the field and, if not, determine how his tests differ so that they can be updated. We present such an approach in the next section.

Using this approach, Jack is shown groups of users whose behaviour best explains the differences between his test workloads and the field. In addition, Jack is also shown key execution events that best explain the differences between each of these groups of users. Jack then discovers a group of users who are using the high-definition group chat feature (i.e., a memory-intensive feature) more strenuously than in the past. Finally, Jack is able to update his test to better reflect the users' changing feature preferences and hence, the system's behaviour in the field.

3 Approach

This section outlines our approach for validating performance tests by automatically deriving workload signatures from execution logs and comparing the signatures from a test against the signatures from the field. Figure 1 provides an overview of our approach. First, we group execution events from the test logs and field logs into workload signatures that describe the workloads. Second, we cluster the workload signatures into groups where a similar set of execution events have occurred. Finally, we analyze the clusters to identify the execution events that correspond to meaningful differences between the performance test and the field. We will describe each phase in detail and demonstrate our approach with a working example of a hypothetical chat application.

3.1 Execution logs

Execution logs record notable events at runtime and are used by developers (to debug a system) and operators (to monitor the operation of a system). They are generated by

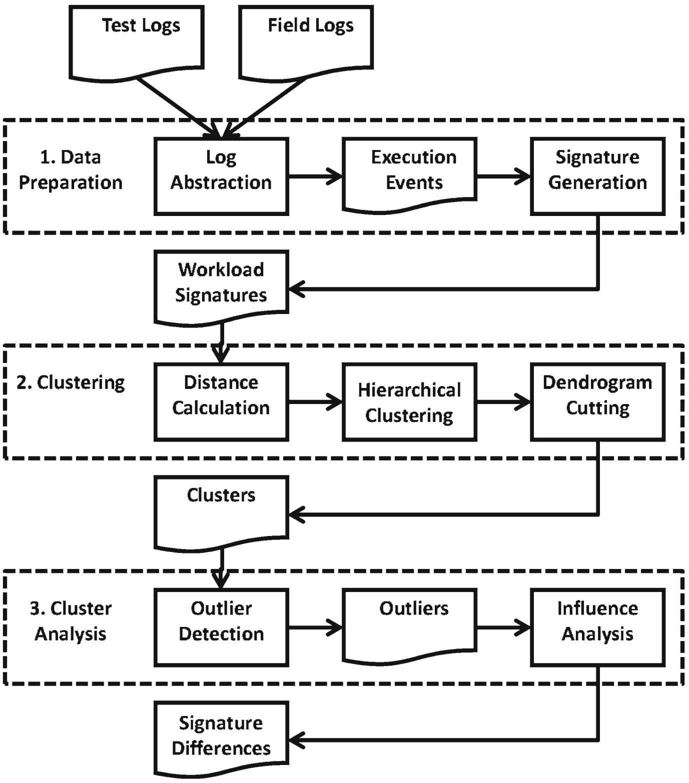

Fig. 1 An overview of our approach

output statements that developers insert into the source code of the system. These output statements are triggered by specific events (e.g., starting, queueing or completing a job) and errors within the system. Compared with performance counters, which usually require explicit monitoring tools (e.g., PerfMon 2014) to be collected, execution logs are readily available in most large-scale systems to support remote issue resolution and legal compliance. For example, the Sarbanes-Oxley Act (The Sarbanes-Oxley Act 2014) requires logging in telecommunication and financial systems.

The second column of Tables 1 and 2 presents the execution logs from our working example. These execution logs contain both static information (e.g., `starts a chat`) and dynamic information (e.g., `Alice` and `Bob`) that changes with each occurrence of an event. Tables 1 and 2 present the execution logs from the field and the test respectively. The test has been configured with a simple use case (from 00:01 to 00:06) that is continuously repeated.

3.2 Data preparation

Execution logs are difficult to analyze because they are unstructured. Therefore, we abstract the execution logs to execution events to enable automated statistical analysis. We then generate workload signatures that represent the behaviour of the system's users.

3.2.1 Log abstraction

Execution logs are not typically designed for automated analysis (Jiang et al. 2008a). Each occurrence of an execution event results in a slightly different log line, because log lines contain static components as well as dynamic information (which may be different for each occurrence of a particular execution event). Dynamic information includes, but it not limited to, user names, IP addresses, URLs, message contents, job IDs and queue sizes. We must remove this dynamic information from the log lines prior to our analysis in order to identify similar execution events. We refer to the process of identifying and removing dynamic information from a log line as "abstracting" the log line.

Our technique for abstracting log lines recognizes the static and dynamic components of each log line using a technique similar to token-based code clone detection (Jiang et al. 2008a). The dynamic components of each log line are then discarded and replaced with ____ (to indicate that dynamic information was present in the original log line). The remaining static components of the log lines (i.e., the abstracted log line) describe execution events.

In order to verify the correctness of our abstraction, many execution logs and their corresponding execution events have been manually reviewed by multiple, independent system experts.

Tables 1 and 2 present the execution events and execution event IDs (a unique ID automatically assigned to each unique execution event) for the execution logs from the field and from the test in our working example. These tables demonstrate the input (i.e., the log lines) and the output (i.e., the execution events) of the log

Table 1 Abstracting execution logs to execution events: execution logs from the field

Time	User	Log line	Execution event	Execution event ID
00:01	Alice	starts a chat with Bob	starts a chat with ___	1
00:01	Alice	says “hi, are you busy?” to Bob	says ___ to ___	2
00:03	Bob	says “yes” to Alice	says ___ to ___	2
00:05	Charlie	starts a chat with Dan	starts a chat with ___	1
00:05	Charlie	says “do you have files?” to Dan	says ___ to ___	2
00:08	Dan	Initiate file transfer to Charlie	Initiate file transfer to ___	3
00:09	Dan	Initiate file transfer to Charlie	Initiate file transfer to ___	3
00:12	Dan	says “got it?” to Charlie	says ___ to ___	2
00:14	Charlie	says “thanks” to Dan	says ___ to ___	2
00:14	Charlie	ends the chat with Dan	ends the chat with ___	4
00:18	Alice	says “ok, bye” to Bob	says ___ to ___	2
00:18	Alice	ends the chat with Bob	ends the chat with ___	4

Table 2 Abstracting execution logs to execution events: execution logs from a performance test

Time	User	Log line	Execution event	Execution event ID
00:01	USER1	starts a chat with USER2	starts a chat with ___	1
00:02	USER1	says “MSG1” to USER2	says ___ to ___	2
00:03	USER2	says “MSG2” to USER1	says ___ to ___	2
00:04	USER1	says “MSG3” to USER2	says ___ to ___	2
00:06	USER1	ends the chat with USER2	ends the chat with ___	4
00:07	USER3	starts a chat with USER4	starts a chat with ___	1
00:08	USER3	says “MSG1” to USER4	says ___ to ___	2
00:09	USER4	ays “MSG2” to USER3	says ___ to ___	2
00:10	USER3	says “MSG3” to USER4	says ___ to ___	2
00:12	USER3	ends the chat with USER4	ends the chat with ___	4
00:13	USER5	starts a chat with USER6	starts a chat with ___	1
00:14	USER5	says “MSG1” to USER6	says ___ to ___	2
00:15	USER6	says “MSG2” to USER5	says ___ to ___	2
00:16	USER5	says “MSG3” to USER6	says ___ to ___	2
00:18	USER5	ends the chat with USER6	ends the chat with ___	4

abstraction process. For example, the `starts a chat with Bob` and `starts a chat with Dan` log lines are both abstracted to the `starts a chat with` ____ execution event.

3.2.2 Signature generation

We generate workload signatures that characterize user behaviour in terms of feature usage expressed by the execution events. In our approach, a workload signature represents either (1) the behaviour of one of the system's users, or (2) the aggregated behaviour of all of the system's users at one point in time. We use the term "user" to describe any type of end user, whether a human or software agent. For example, the end users of a system such as Amazon.com are both human and software agents (e.g., "shopping bots" that search multiple websites for the best prices). Workload signatures are represented as points in an n-dimensional space (where n is the number of unique execution events).

Workload signatures representing individual users are generated for each user because workloads are driven by the behaviour of the system's users. We have also found cases when an execution event only causes errors when over-stressed by an individual user (i.e., one user executing the event 1,000 times has a different impact on the system's behaviour than 100 users each executing the event 10 times) (Syer et al. 2014). Therefore, it is important to identify users whose behaviour is seen in the field, but not during the test.

Workload signatures representing individual users are generated in two steps. First, we identify all of the unique user IDs that appear in the execution logs. Users represent a logical "unit of work" where a workload is the sum of one or more units of work. In systems primarily used by human end users (e.g., e-commerce and telecommunications system), user IDs may include user names, email addresses or device IDs. In systems primarily used for processing large amounts of data (e.g., distributed data processing frameworks such as Hadoop), user IDs may include job IDs or thread IDs. The second column of Table 3 presents all of the unique user IDs identified from the execution logs of our working example. Second, we generate a signature for each user ID by counting the number of times that each type of execution event is attributable to each user ID. For example, from Table 1, we see that `Alice` starts one chat, sends two messages and ends one chat. Table 3 shows the signatures generated for each user using the events in Tables 1 and 2.

Workload signatures representing the aggregated users are generated for short periods of time (e.g., 1 min) to represent the traditional notion of a "workload" (i.e., the total number and mix of incoming requests to the system). The system's resource usage is highly dependent on these workloads. Unlike the workload signatures representing individual users, the workload signatures representing aggregated users capture the "burstiness" (i.e., the changes in the number of request per seconds) of the workload. Therefore, it is important to identify whether the the aggregated user behaviour that is seen in the field is also seen during the test.

Workload signatures representing the aggregated users are generated by grouping the execution logs into time intervals (i.e., grouping the execution logs that occur between two points in time). Grouping is a two step process. First, we specify the

Table 3 Workload signatures representing individual users

	User ID	Execution event ID			
		1	2	3	4
		`start chat`	`send message`	`transfer file`	`end chat`
Field users	`Alice`	1	2	0	1
	`Bob`	0	1	0	0
	`Charlie`	1	2	0	1
	`Dan`	0	1	2	0
Test users	`USER1`	1	2	0	1
	`USER2`	0	1	0	0
	`USER3`	1	2	0	1
	`USER4`	0	1	0	0
	`USER5`	1	2	0	1
	`USER6`	0	1	0	0

Table 4 Workload signatures representing the aggregated users

	Time	Execution event ID			
		1	2	3	4
		`start chat`	`send message`	`transfer file`	`end chat`
Field times	`00:01-00:06`	2	3	0	0
	`00:07-00:12`	0	1	2	0
	`00:13-00:18`	0	2	0	2
Test times	`00:01-00:06`	1	3	0	1
	`00:07-00:12`	1	3	0	1
	`00:13-00:18`	1	3	0	1

length of the time interval. In our previous work, we found that time intervals of 90–150 s perform well when generating workload signatures that represent the aggregated user behaviour (Syer et al. 2013). However, these time intervals may vary between systems. System experts should determine the optimal time interval (i.e., a time interval that provides the necessary detail without an unnecessary overhead) for their systems. Alternatively, system experts may specify multiple time intervals and generate overlapping signatures (e.g., generating signatures representing the aggregated user behaviour in 1, 3 and 5 min time intervals). Second, we generate a signature for each time interval by counting the number of times that each type of execution event occurs in that time interval. For example, from Table 1, we see that one chat is started and three messages are sent between time `00:01` and `00:06`. Table 4 shows the signatures generated for each 6 s time interval using the events in Tables 1 and 2. From Table 4, we see that all three signatures generated from the test are identical. This is to be expected because the test was configured with a simple use case (from 00:01 to 00:06) that is continuously repeated.

Our approach considers the individual user signatures and the aggregate user signatures separately. Therefore, the Clustering and Cluster Analysis phases are applied once to the individual user signatures and once to the aggregate user signatures. For brevity, we will demonstrate the remainder of our approach using only the individual user signatures in Table 3.

3.3 Clustering

The second phase of our approach is to cluster the workload signatures into groups where a similar set of events have occurred. We can then identify groups of similar, but not necessary identical, workload signatures.

The clustering phase in our approach consists of three steps. First, we calculate the dissimilarity (i.e., distance) between every pair of workload signatures. Second, we use a hierarchical clustering procedure to cluster the workload signatures into groups where a similar set of events have occurred. Third, we convert the hierarchical clustering into k partitional clusters (i.e., where each workload signature is a member in only one cluster). We have automated the clustering phase using scalable statistical techniques.

3.3.1 Distance calculation

Each workload signature is represented by one point in an n-dimensional space (where n is the number of unique execution events). Clustering procedures rely on identifying points that are "close" in this n-dimensional space. Therefore, we must specify how distance is measured in this space. A larger distance between two points implies a greater dissimilarity between the workload signatures that these points represent. We calculate the distance between every pair of workload signatures to produce a distance matrix.

We use the Pearson distance, a transform of the Pearson correlation (Fulekar 2008), as opposed to the many other distance measures (Fulekar 2008; Cha 2007; Frades and Matthiesen 2009), as the Pearson distance often produces a clustering that is a closer match to the manually assigned clusters (Sandhya and Govardhan 2012; Huang 2008). We find that the Pearson distance performs well when clustering workload signatures (see Sect. 5.3; Syer et al. 2014).

We first calculate the Pearson correlation (ρ) between two workload signatures using Eq. 1. This measure ranges from -1 to $+1$, where a value of 1 indicates that the two workload signatures are identical, a value of 0 indicates that there is no relationship between the signatures and a value of -1 indicates an inverse relationship between the signatures (i.e., as the occurrence of specific execution events increase in one workload signature, they decrease in the other).

$$\rho = \frac{n \sum_i^n x_i \times y_i - \sum_i^n x_i \times \sum_i^n y_i}{\sqrt{\left(n \sum_i^n x_i^2 - \left(\sum_i^n x_i\right)^2\right) \times \left(n \sum_i^n y_i^2 - \left(\sum_i^n y_i\right)^2\right)}} \quad (1)$$

where x and y are two workload signatures and n is the number of execution events.

Table 5 Distance matrix

	Alice	Bob	Charlie	Dan	USER1	USER2	USER3	USER4	USER5	USER6
Alice	0	0.184	0	0.426	0	0.184	0	0.184	0	0.184
Bob	0.184	0	0.184	0.826	0.184	0	0.184	0	0.184	0
Charlie	0	0.184	0	0.426	0	0.184	0	0.184	0.	0.184
Dan	0.426	0.826	0.426	0	0.426	0.826	0.426	0.826	0.426	0.826
USER1	0	0.184	0	0.426	0	0.184	0	0.184	0	0.184
USER2	0.184	0	0.184	0.826	0.184	0	0.184	0	0.184	0
USER3	0	0.184	0	0.426	0	0.184	0	0.184	0	0.184
USER4	0.184	0	0.184	0.826	0.184	0	0.184	0	0.184	0
USER5	0	0.184	0	0.426	0	0.184	0	0.184	0	0.184
USER6	0.184	0	0.184	0.826	0.184	0	0.184	0	0.184	0

We then transform the Pearson correlation (ρ) to the Pearson distance (d_ρ) using Eq. 2.

$$d_\rho = \begin{cases} 1 - \rho & \text{for} \quad \rho \geq 0 \\ |\rho| & \text{for} \quad \rho < 0 \end{cases} \tag{2}$$

Table 5 presents the distance matrix produced by calculating the Pearson distance between every pair of workload signatures in our working example.

3.3.2 Hierarchical clustering

We use an agglomerative, hierarchical clustering procedure (Tan et al. 2005) to cluster the workload signatures using the distance matrix calculated in the previous step. The clustering procedure starts with each signature in its own cluster and proceeds to find and merge the closest pair of clusters (using the distance matrix), until only one cluster (containing everything) is left. One advantage of hierarchical clustering is that we do not need to specify the number of clusters prior to performing the clustering. Further, performance analysts can change the number of clusters (e.g., to produce a larger number of more cohesive clusters) without having to rerun the clustering phase.

Hierarchical clustering updates the distance matrix based on a specified linkage criterion. We use the average linkage, as opposed to the many other linkage criteria (Frades and Matthiesen 2009; Tan et al. 2005), as the average linkage is the de facto standard (Frades and Matthiesen 2009; Tan et al. 2005). The average linkage criterion is also the most appropriate when little information about the expected clustering (e.g., the relative size of the expected clusters) is available. We find that the average linkage criterion performs well when clustering workload signatures (see Sect. 5.3; Syer et al. 2014).

When two clusters are merged, the average linkage criterion updates the distance matrix in two steps. First, the merged clusters are removed from the distance matrix. Second, a new cluster (containing the merged clusters) is added to the dis-

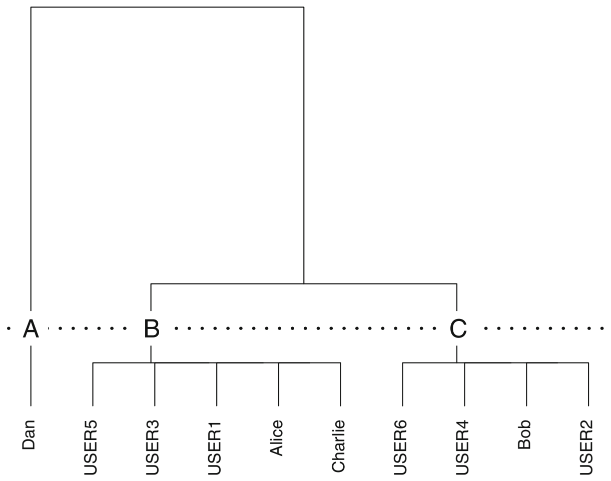

Fig. 2 Sample dendrogram. The *dotted horizontal line* indicates where the dendrogram was cut into three clusters (i.e., Cluster A, B and C)

tance matrix by calculating the distance between the new cluster and all existing clusters. The distance between two clusters is the average distance (as calculated by the Pearson distance) between the workload signatures of the first cluster and the workload signatures of the second cluster (Frades and Matthiesen 2009; Tan et al. 2005).

We calculate the distance between two clusters ($d_{x,y}$) using Eq. 3.

$$d_{x,y} = \frac{1}{n_x \times n_y} \times \sum_{i}^{n_x} \sum_{j}^{n_y} d_\rho(x_i, y_j) \tag{3}$$

where $d_{x,y}$ is the distance between cluster x and cluster y, n_x is the number of workload signatures in cluster x, n_y is the number of workload signatures in cluster y and $d_\rho(x_i, y_j)$ is the Pearson distance between workload signature i in cluster x and workload signature j in cluster y.

Figure 2 shows the dendrogram produced by hierarchically clustering the workload signatures from our working example.

3.3.3 Dendrogram cutting

The result of a hierarchical clustering procedure is a hierarchy of clusters. This hierarchy is typically visualized using hierarchical cluster dendrograms. Figure 2 is an example of a hierarchical cluster dendrogram. Such dendrograms are binary tree-like diagrams that show each stage of the clustering procedure as nested clusters (Tan et al. 2005).

To complete the clustering procedure, the dendrogram must be cut at some height. This height represents the maximum amount of intra-cluster dissimilarity that will be accepted within a cluster before that cluster is further divided. Cutting the dendrogram results in a clustering where each workload signature is assigned to only one cluster. Such a cutting of the dendrogram is done either by (1) manual (visual) inspection or (2) statistical tests (referred to as stopping rules).

Although a visual inspection of the dendrogram is flexible and fast, it is subject to human bias and may not be reliable. We use the Calinski–Harabasz stopping rule (Calinski and Harabasz 1974), as opposed to the many other stopping rules (Calinski and Harabasz 1974; Duda and Hart 1973; Milligan and Cooper 1985; Mojena 1977; Rousseeuw 1987), as the Calinski–Harabasz stopping rule most often cuts the dendrogram into the correct number of clusters (Milligan and Cooper 1985). We find that the Calinski–Harabasz stopping rule performs well when cutting dendrograms produced by clustering workload signatures (see Sect. 5.3; Syer et al. 2014).

The Calinski–Harabasz stopping rule is a pseudo-*F-statistic*, which is a ratio reflecting within-cluster similarity and between-cluster dissimilarity. The optimal clustering will have high within-cluster similarity (i.e., the workload signatures within a cluster are similar) and a high between-cluster dissimilarity (i.e., the workload signatures from two different clusters are dissimilar).

The dotted horizontal line in Fig. 2 shows where the Calinski–Harabasz stopping rule cut the hierarchical cluster dendrogram from our working example into three clusters (i.e., the dotted horizontal line intersects with solid vertical lines at three points in the dendrogram). Cluster A contains one user (Dan), cluster B contains four users (Alice, Charlie, USER1 and USER3) and cluster C contains three users (Bob, USER2 and USER4).

3.4 Cluster analysis

The third phase in our approach is to identify the execution events that correspond to the differences between the workload signatures from the performance test and the field. As execution logs may contain billions of events describing the behaviour of millions of users, this phase will only identify the most important workload signature differences. Therefore, our approach helps system experts to update their performance tests by identifying the most meaningful differences between their performance tests and the field. Such "filtering" provides performance analysts with a concrete list of events to investigate.

The cluster analysis phase of our approach consists of two steps. First, we detect outlying clusters. Outlying clusters contain workload signatures that are not well represented in the test (i.e., workload signatures that occur in the field significantly more than in the test). Second, we identify key execution events of the outlying clusters. We refer to these execution events as "signature differences". Knowledge of these signature differences may lead performance analysts to update their performance tests. "Event A occurs 10 % less often in the test relative to the field" is an example of a signature difference that may lead performance analysts to update a test such that Event A occurs more frequently. We use scalable statistical techniques to automate this step.

3.4.1 Outlying cluster detection

Clusters contain workload signatures from the field and/or the test. When clustering workload signatures from a field-representative performance test and the field, we would expect that each cluster would have the same proportion of workload signatures from the field compared to workload signatures from the test. Clusters with a high proportion of workload signatures from the field relative to the test would then be considered "outlying" clusters. These outlying clusters contain workload signatures that represent behaviour that is seen in the field, but not during the test.

We identify outlying clusters using a *one-sample upper-tailed z-test for a population proportion*. These tests are used to determine whether the observed sample proportion is significantly larger than the hypothesized population proportion. The difference between the observed sample proportion and the hypothesized population proportion is captured by a *z-score* (Sokal and Rohlf 2011). Higher *z-scores* indicate an increased probability that the observed sample proportion is greater than the hypothesized population proportion (i.e., that the cluster contains a greater proportion of workload signatures from the field). Hence, as the *z-score* of a particular cluster increases, the probability that the cluster is an outlying cluster also increases. *One-sample z-tests for a proportion* have successfully been used to identify outliers in software engineering data using these hypotheses (Kremenek and Engler 2003; Jiang et al. 2008b; Syer et al. 2014).

We construct the following hypotheses to be tested by a *one-sample upper-tailed z-test*. Our *null hypothesis* assumes that the proportion of workload signatures from the field in a cluster is less than 90 %. Our *alternate hypothesis* assumes that this proportion is greater than 90 %.

Equation 4 presents how the *z-score* of a particular cluster is calculated.

$$p = \frac{n_x}{n_x + n_y} \tag{4}$$

$$\sigma = \sqrt{\frac{p_0 \times (1 - p_0)}{n_x + n_y}} \tag{5}$$

$$z = \frac{p - p_0}{\sigma} \tag{6}$$

where n_x is the number of workload signatures from the field in the cluster, n_y is the number of workload signatures from the test in the cluster, p is the proportion of workload signatures from the field in the cluster, σ is the standard error of the sampling distribution of p and p_0 is the hypothesized population proportion (i.e., 90 %, the null hypothesis).

We then use the *z-score* to calculate a *p-value* to determine whether the sample population proportion is significantly greater than the hypothesized proportion population. This *p-value* accounts for differences in the total number of workload signatures from the test compared to the field as well as variability in the proportion of workload signatures from the field across the clusters.

Equation 7 presents how the *p-value* of a particular cluster is calculated.

Table 6 Identifying Outlying Clusters

Cluster	Size	# Signatures from:		*z-score*	*p-value*
		Field	Test		
A	1	1	0	0.333	0.68
B	5	2	3	−3.737	1.00
C	4	1	3	−4.333	1.00

$$Z(x, \mu, \sigma) = \frac{1}{\sigma \times \sqrt{2 * \pi}} \times e^{\frac{-(x-\mu)^2}{2\times\sigma^2}} \tag{7}$$

$$p = P(Z > z) \tag{8}$$

where μ is the average proportion of workload signatures in a cluster, σ is the standard deviation of the proportion of workload signatures in a cluster, $Z(x, \mu, \sigma)$ is the normal distribution given μ and σ and p is the *p-value* of the test.

Table 6 presents the size (i.e., the number of workload signatures in the cluster), breakdown (i.e., the number of workload signatures from the performance test and the field), *z-score* and *p-value* for each cluster in our working example (i.e., each of the clusters that were identified when the Calinski–Harabasz stopping rule was used to cut the dendrogram in Fig. 2).

From Table 6, we find that clusters with a greater proportion of workload signatures from the field have a larger *z-score*. For example, the proportion of workload signatures from the field in Cluster A is 100 % and the *z-score* is 0.333, whereas the proportion of workload signatures from the field in Cluster B is 40 % (i.e., 2/5) and the *z-score* is -3.737.

From Table 6, we also find that the proportion of workload signatures from the field in any one cluster is not significantly more than 90 % (i.e., no *p-values* are less than 0.05). Therefore, no clusters are identified as outliers. However, outliers are extremely difficult to detect in such small data sets. Therefore, for the purposes of this working example, we will assume that Cluster A has been identified as an outlier because its *z-score* (0.333) is much larger than the *z-scores* of Cluster B (−3.737) or Cluster C (−4.333).

3.4.2 Signature difference detection

We identify the differences between workload signatures in outlying clusters and the average ("normal") workload signature using statistical measures (i.e., *unpaired two-sample two-tailed Welch's unequal variances t-tests* (Student 1908; Welch 1997) and *Cohen's d* effect size (Cohen 1988)). This analysis quantifies the importance of each execution event in differentiating a cluster. Knowledge of these events may lead performance analysts to update their tests.

First, we determine the execution events that differ significantly between the workload signatures in the outlying clusters and the average workload signature. For example, execution events that occur 10 times more often in the workload signatures of an outlying cluster compared to the average workload signature should likely be flagged for further analysis by a system expert.

We perform an *unpaired two-sample two-tailed Welch's unequal variances t-test* to determine which execution events differ significantly between the workload signatures in an outlying cluster and the average workload signature. These tests are used to determine whether the difference between two population means is statistically significant (Student 1908; Welch 1997). The difference between the two population means is captured by a *t-statistic*. Larger absolute *t-statistics* (i.e., the absolute value of the *t-statistic*) indicate an increased probability that the two population means differ (i.e., that the number of times an execution event occurs in the workload signatures of an outlying cluster compared to the average workload signature differs). Hence, as the absolute value of the *t-statistic* of a particular execution event and outlying cluster increases, the probability that the number of times an execution event occurs in the workload signatures of an outlying cluster compared to the average workload signature also increases. *T-tests* are one of the most frequently performed statistical tests (Elliott 2006).

We construct the following hypotheses to be tested by an *unpaired two-sample two-tailed Welch's unequal variances t-test*. Our *null hypothesis* assumes that an execution event occurs the same number of times in the workload signatures of an outlying cluster compared to the average workload signature. Conversely, our *alternate hypothesis* assumes that the execution event does not occur the same number of times in an outlying cluster compared to the average workload signature.

Equation 9 presents how the *t-statistic* for a particular execution event and a particular outlying cluster is calculated.

$$\sigma = \sqrt{\frac{(n_x - 1) \times \sigma_x^2 + (n_y - 1) \times \sigma_y^2}{n_x + n_y + 2}} \tag{9}$$

$$t = \frac{\mu_x - \mu_y}{\sqrt{\frac{\sigma_x^2}{n_x} + \frac{\sigma_x^2}{n_y}}} \tag{10}$$

where n_x is the number of workload signatures in the outlying cluster, n_y is the total number of workload signatures that are not in the outlying cluster, μ_x is the average number of times the execution event occurs in the workload signatures in the outlying cluster, μ_y is the average number of times the execution event occurs in all the workload signatures that are not in the outlying cluster, σ_x is the variance of the number of times the execution event occurs in the workload signatures in the outlying cluster, σ_y is the variance of the number of times the execution event occurs in all of the workload signatures that are not in the outlying cluster, σ is the pooled standard deviation of the number of times the execution event occurs in the workload signatures in the outlying cluster and the number of times the execution event occurs in all the workload signatures and t is the *t-statistic*.

We then use the *t-statistic* to calculate a *p-value* to test whether the difference between the number of times an execution event occurs in the workload signatures of an outlying cluster compared to the average workload signature is statistically significant.

Table 7 Identifying influential execution events

Execution event ID	*t-statistic*	*p-value*	*Cohen's d*
1	0.90	0.39	0.95
2	0.90	0.39	0.95
3	−2.71	0.02	2.85
4	0.90	0.39	0.95

Equation 11 presents how the *p-value* for a particular execution event and a particular outlying cluster is calculated.

$$v = \frac{\frac{\sigma_x^2}{n_x} + \frac{\sigma_x^2}{n_y}}{\frac{\left(\frac{\sigma_x^2}{n_x}\right)^2}{n_x - 1} + \frac{\left(\frac{\sigma_y^2}{n_y}\right)^2}{n_y - 1}} \tag{11}$$

$$T = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\sqrt{v \times \pi} \times \Gamma\left(\frac{v}{2}\right)} \times \left(1 + \frac{x^2}{v}\right)^{-\frac{v+1}{2}} \tag{12}$$

$$p = 2 \times P(T < t) \tag{13}$$

where v is the degree of freedom, Γ is the gamma function, T is the *t-distribution* and p is the *p-value* of the test.

Table 7 shows the *t-statistic* and the associated *p-value* for each execution event in the outlying cluster (i.e., Cluster A).

From Table 7, we find that Execution Event ID 3 (i.e., initiate a file transfer) differs significantly between Cluster A and the average workload signatures (i.e., $p < 0.01$). From the workload signatures in Table 3, we see that Execution Event ID 3 occurs twice in Dan's workload signature (i.e., the only workload signature in Cluster A), but never in the other workload signatures.

Second, we determine the most important execution events that differ between the workload signatures in the outlying clusters and the average workload signature. For example, if execution events "A" and "B" occur 2 and 10 times more often in the workload signatures of an outlying cluster compared to the average workload signature, then execution event "B" should be flagged for further analysis by a system expert rather than execution event "A."

We calculate the *Cohen's d* effect size to determine the most important execution events that differ between the workload signatures in the outlying clusters and the average workload signature. *Cohen's d* effect size measures the difference between two population means (Cohen 1988). Larger *Cohen's d* effect sizes indicate a greater difference between the two population means, regardless of statistical significance. Hence, as the *Cohen's d* effect size of a particular execution event and outlying cluster increases, the difference between the number of times an execution event occurs in the workload signatures of an outlying cluster compared to the average workload signature also increases.

Equation 14 presents how *Cohen's d* is calculated for a particular execution event and a particular outlying cluster.

$$\sigma = \sqrt{\frac{(n_x - 1) \times \sigma_x^2 + (n_y - 1) \times \sigma_y^2}{n_x + n_y + 2}} \tag{14}$$

$$d = \frac{\mu_x - \mu_y}{\sigma} \tag{15}$$

where n_x is the number of workload signatures in the outlying cluster, n_y is the total number of workload signatures that are not in the outlying cluster, μ_x is the average number of times the event occurs in the workload signatures in the outlying cluster, μ_y is the average number of times the event occurs in all the workload signatures that are not in the outlying cluster, σ_x is the variance in the number of times the event occurs in the workload signatures in the outlying cluster, σ_y is the variance in the number of times the event occurs in all of the workload signatures that are not in the outlying cluster, σ is the pooled standard deviation of the number of times the event occurs in the workload signatures in the outlying cluster and the number of times the event occurs in all the workload signatures and d is *Cohen's d*.

Cohen's d effect size is traditionally interpreted as follows:

$$\begin{cases} \text{trivial} & \text{for } d < 0.2 \\ \text{small} & \text{for } 0.2 < d \leq 0.5 \\ \text{medium} & \text{for } 0.5 < d \leq 0.8 \\ \text{large} & \text{for } d > 0.8 \end{cases} \tag{16}$$

However, such an interpretation was originally proposed for the social sciences. Kampenes et al. (2007) performed a systematic review of 103 software engineering papers empirically established the following interpretation of *Cohen's d* effect size in software engineering.

$$\begin{cases} \text{trivial} & \text{for } d < 0.17 \\ \text{small} & \text{for } 0.17 < d \leq 0.6 \\ \text{medium} & \text{for } 0.6 < d \leq 1.4 \\ \text{large} & \text{for } d > 1.4 \end{cases} \tag{17}$$

From Table 7, we find that Execution Event ID 3 (i.e., initiate a file transfer) has a large (i.e., $d > 1.4$) effect size indicating that the difference in Execution Event ID 3 between the workload signatures in Cluster A and the average workload signature is large.

Finally, we identify the influential events as any execution event with a *t-test p-value* less than 0.01 and a *Cohen's d* greater than 1.4. Table 7 shows the *Cohen's d* for each execution event in the outlying cluster (i.e., Cluster A).

From Table 7, we find that the difference in Execution Event ID 3 between the workload signatures in Cluster A and the average workload signature is statistically significant (i.e., $p < 0.01$) and large (i.e., $d > 1.4$). Therefore, our approach identifies one workload signature (i.e., the workload signature representing the user Dan) as a key difference between the test and the field of our working example. In particular, we identify one execution event (i.e., initiate a file transfer) that is not well represented in the test (in fact it does not occur at all). Performance analysts should then adjust the workload intensity of the file transfer functionality in the test.

In our simple working example, performance analysts could have examined how many times each execution event had occurred and identified events that occur much more frequently in the field compared to the test. However, in practice, data sets are considerably larger. For example, our enterprise case studies contain over hundreds of different types of execution events and millions of log lines. Further, some execution events have a different impact on the system's behaviour based on the manner in which the event is executed. For example, our second enterprise case study identifies an execution event that only causes errors when over-stressed by an individual user (i.e., one user executing the event 1,000 times has a different impact on the system's behaviour than 100 users each executing the event 10 times). Therefore, in practice performance analysts cannot simply examine occurrence frequencies.

4 Case studies

This section outlines the setup and results of our case studies. First, we present two case studies using a Hadoop application. We then discuss the results of three case studies using an enterprise system. Table 8 outlines the systems and data sets used in our case studies.

Our case studies aim to determine whether our approach can detect workload signature differences due to feature, intensity and issue differences between a performance test and the field. Our case studies include systems whose users are either human (Enterprise System) or software (Hadoop) agents.

We compare our results with our previous approach to comparing workload signatures. We empirically evaluate the improvement in our ability to flag execution events that best describe the differences between a performance test and the field (Syer et al. 2014).

We also compare our results with the current state-of-the-practice. Currently, performance analysts validate performance tests by comparing the number of times each execution event has occurred during the test compared to the field and investigating any differences. Therefore, we rank the events based on the difference in occurrence between the test and the field. We then investigate the events with the largest differences. In practice, performance analysts do not know how many of these events should be investigated. Therefore, we examine the same number of events as our approach identifies such that the effort required by performance analysts to manually analyze the events flagged by either (1) our approach or (2) the state of the practice is approximately equal. For example, if our approach flags 10 execution events, we examine the top 10 events ranked by the state-of-the-practice. We then compare the precision of our approach to the state-of-the-pratice. We define precision as the percentage of execution events that our approach identified as meaningful differences between the system's behaviour in the field and during the test that multiple, independent system

Table 8 Case study subject systems

	Hadoop			Enterprise system		
Application domain:	Data processing			Telecom		
License:	Open-source			Enterprise		
Performance test data						
# Log lines	3,862	6,851	169,627	9,295,418	6,788,510	2,341,174
Notes	Performance test driven by the Hadoop WordCount application	Performance test driven by the Hadoop WordCount application	Performance test driven by the Hadoop Exotic Songs application	Use-case performance test driven by a load generator	Performance test driven by a replay script	Field-representative performance test driven by a replay script
Field data						
# Log lines	6,120	45,262	173,235	6,788,510	7,383,738	2,517,558
Notes	The system experienced a machine failure in the field	The system experienced a Java heap space exception in the field	The system experienced a performance degradation in the field	System experts confirmed that there were no errors in the field	The system experienced a crash in the field	System experts confirmed that there were no errors in the field
Case study						
Case study name	Machine failure in the field	Java heap space error in the field	LZO Compression enabled in the field	Comparing use-case performance tests to the field	Comparing replay performance tests to the field	Comparing field-representative replay performance tests to the field
Type of differences	Issue difference	Issue difference	Feature difference	Intensity and feature differences	Intensity difference	No difference

Table 8 continued

Application domain	Hadoop			Enterprise system		
	Data processing			Telecom		
License	Open-source			Enterprise		
Results						
Influential events	12	3	2	28	5	0
Precision (our approach)	91.7 %	66.7 %	100 %	92.9 %	100 %	100 %[a]
Precision (our previous approach)	100 %	25 %	0 %	26.9 %	100 %	0 %
Precision (state-of-the-practice)	58.3 %	66.7 %	100 %	42.9 %	0 %	100 %[a]
Precision (statistical comparison)	0 %	0 %	100 %	14.9 %	0 %	0 %

[a] No execution events were flagged because the field and the test do not differ

experts confirmed are meaningful differences between the system's behaviour in the field and during the test.

We also compare our results to the results of a basic statistical comparison of the execution logs from a test and the field. We use the same statistical measures outlined in Sect. 3.4.2 (i.e., *t-tests* and *Cohen's d*) to statistically compare the number of times each execution event has occurred during the test compared to the field. This statistical comparison is identical to our approach when one workload signature representing the aggregated user behaviour is generated from the test and another from the field (i.e., our approach without clustering). We flag all events with a statistically significant (i.e., $p < 0.01$) and large (i.e., $d > 1.4$) difference between the test and the field. This comparison demonstrates the value added by our approach, specifically in generating workload signatures that represent the behaviour of the system's users, compared to a simple statistical comparison of the execution logs.

4.1 Hadoop case study

4.1.1 The Hadoop platform

Our first case study system are two applications that are built on the Hadoop platform. Hadoop is an open-source distributed data processing platform that implements MapReduce (Hadoop 2014; Dean and Ghemawat 2008).

MapReduce is a distributed data processing framework that allows large amounts of data to be processed in parallel by the nodes of a distributed cluster of machines (Dean and Ghemawat 2008). The MapReduce framework consists of two steps: a Map step, where the input data is divided amongst the nodes of the cluster, and a Reduce step, where the results from each of the nodes is collected and combined.

Operationally, a Hadoop application may contain one or more MapReduce steps (each step is a "Job"). Jobs are further broken down into "tasks," where each task is either a Map task or a Reduce task. Finally, each task may be executed more than once to support fault tolerance within Hadoop (each execution is an "attempt").

4.1.2 The WordCount application

The first Hadoop application used in this case study is the WordCount application (MapReduce Tutorial 2014). The WordCount application is a standard example of a Hadoop application that is used to demonstrate the Hadoop platform and the MapReduce framework. The WordCount application reads one or more text files (a corpus) and counts the number of times each unique word occurs within the corpus.

4.1.2.1 Machine failure in the field We monitored the performance of the Hadoop WordCount application during a performance test. The performance test workload consisted of 3.69 gigabytes of text files (i.e., the WordCount application counts the number of times each unique word occurs in these text files). The cluster contains five machines, each with dual Intel Xeon E5540 (2.53 GHz) quad-core CPUs, 12 GB memory, a Gigabit network adaptor and SATA hard drives. While this cluster is small by industry standards (Chen et al. 2012), recent research has shown that almost all failures can be reproduced on three machines (Yuan et al. 2014).

We then monitored the performance of the Hadoop WordCount application in the field and found that the performance was much less than expected based on our performance tests. We found that the throughput (completed attempts/s) was much lower than the throughput achieved during testing and that the average network IO (bytes/s transfered between the nodes of the cluster) was considerably lower than the average historical network IO. Therefore, we compare the execution logs from the field and the test to determine whether our tests accurately represent the current conditions in the field.

We apply our approach to the execution logs collected from the WordCount application in the field and during the test. We generate a workload signature for each attempt because these attempts are the "users" of the Hadoop platform. These workload signatures represent the individual user behaviour discussed in Sect. 3.2.2. We also generate workload signatures for each 1, 3 and 5 min time interval. These workload signatures represent the aggregated user behaviour discussed in Sect. 3.2.2. Our approach identifies 12 workload signature differences (i.e., execution events that best describe the differences between the field and the test) for analysis by system experts. We only report a selection of these execution events here for brevity.

```
INFO org.apache.hadoop.hdfs.DFSClient: Abandoning block blk_id

INFO org.apache.hadoop.hdfs.DFSClient: Exception in
createBlockOutputStream java.io.IOException: Bad connect ack with
firstBadLink ip_address

WARN org.apache.hadoop.hdfs.DFSClient: Could not get block
locations. Source file - Aborting...

INFO org.apache.hadoop.mapred.TaskRunner: Runnning cleanup
for the task
```

These execution events indicate that the WordCount application (1) cannot retrieve data from the Hadoop File System (HFS), (2) has a "bad" connection with the node at `ip_address` and (3) cannot reconnect to the datanode (datanodes store data in the HFS) at `ip_address`. The remaining execution events are warning messages associated with this error. Made aware of this issue, performance analysts could update their performance tests to test how the system responds to machine failures and propose redundancy in the field.

The last execution event is a clean-up event (e.g., removing temporary output directories after the job completes)(OutputCommitter 2014). This execution event occurs more frequently in the field compared to the test because a clean-up is always run after an attempt fails (MapReduce Tutorial 2014). However, system experts do not believe that this is a meaningful difference between the system's behaviour in the field and during the test. Hence, we have correctly identified 11 events out of the 12 flagged events. The precision of our approach is 91.7 %.

To empirically evaluate the improvement over our previous approach, we use our previous approach (Syer et al. 2014) to identify the execution events that best explain the differences between the system's behaviour during a performance test and in the field. Our previous approach identifies the following 3 workload signature differences:

```
INFO org.apache.hadoop.hdfs.DFSClient: Abandoning block blk_id

INFO org.apache.hadoop.hdfs.DFSClient: Exception in
createBlockOutputStream java.io.IOException: Bad connect ack with
firstBadLink ip_address

INFO org.apache.hadoop.ipc.Client: Retrying connect to server:
ip_address. Already tried NUMBER time(s)
```

All of these workload signature differences describes meaningful differences between the system's behaviour in the field and during the test. In addition, most of these differences are also identified by our new approach. Therefore, the precision of our previous approach is 100 % (i.e., 3/3). However, our previous approach only identifies 3 differences whereas our new approach correctly identifies 11 differences. Therefore, the recall of our previous approach is lower than our new approach.

We also use the state-of-the-practice approach (outlined in Sect. 4) to identify the execution events with the largest occurrence frequency difference between the field and the test. We examine the top 12 execution events ranked by largest difference in occurrence in the field compared to the test. We find that 7 of these events describe important differences between the field and the test (all of these events were found by our approach). However, 5 of these events do not describe important differences between the field and the test (e.g., the clean-up or a start up event such as initializing JVM metrics (Metrics 2.0 2014)). Therefore, the precision of the state-of-the-practice is 58.3 % (i.e., 7/12). We also use a statistical comparison (outlined in Sect. 4) to identify the execution events that differ between the field and the test. However, no execution events were flagged using this method.

4.1.2.2 Java heap space error in the field We monitored the performance of the Hadoop WordCount application during a performance test. The performance test workload consisted of 15 gigabytes of text files.

We then monitored the Hadoop WordCount application in the field and found that the throughput (completed attempts/s) was much lower than the throughput achieved during testing. We also found that the ratio of completed to failed attempts was much lower (i.e., more failed attempts relative to completed attempts) in the field compared to our performance test. Therefore, we compare the execution logs from the test and the field to determine whether our tests accurately represent the current conditions in the field.

Our approach identifies the following 3 workload signature differences:

```
FATAL org.apache.hadoop.mapred.Child: Error running child :
java.lang.OutOfMemoryError: Java heap space
   at org.apache.hadoop.io.Text.setCapacity(Text.java:240)
   at org.apache.hadoop.io.Text.append(Text.java:216)
   at org.apache.hadoop.util.LineReader.readLine
   (LineReader.java:159)
   at org.apache.hadoop.mapred.LineRecordReader.next
   (LineRecordReader.java:133)
   at org.apache.hadoop.mapred.LineRecordReader.next
   (LineRecordReader.java:38)
   at org.apache.hadoop.mapred.MapTask$TrackedRecordReader.moveToNext
   (MapTask.java:236)
   at org.apache.hadoop.mapred.MapTask$TrackedRecordReader.next
   (MapTask.java:216)
   at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:48)
   at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:436)
   at org.apache.hadoop.mapred.MapTask.run(MapTask.java:372)
   at org.apache.hadoop.mapred.Child$4.run(Child.java:255)
   at java.security.AccessController.doPrivileged(Native Method)
   at javax.security.auth.Subject.doAs(Subject.java:415)
   at org.apache.hadoop.security.UserGroupInformation.doAs
   (UserGroupInformation.java:1121)
   at org.apache.hadoop.mapred.Child.main(Child.java:249)

INFO org.apache.hadoop.mapred.Task: Aborting job with runstate : FAILED

INFO org.apache.hadoop.mapred.Task: Cleaning up job
```

These execution events indicate that the WordCount application (1) suffers a `java.lang.OutOfMemoryError` and (2) the `java.lang.OutOfMemoryError` causes attempts to fail. When performance analysts consult with the official Hadoop documentation, they find that input files are split using line-feeds or carriage-returns (TextInputFormat 2014). Further, when performance analysts examine the input files that the Hadoop WordCount application fails to process, they find that these files lack line-feeds or carriage-returns due to a conversion error between DOS and UNIX. Made aware of this issue, performance analysts could configure a maximum line size using RecordReader (RecordReader 2014) to prevent this error in the field.

As before, the last execution event is a clean-up event that system experts do not believe is a meaningful difference between the system's behaviour in the field and during the test. Hence, we have correctly identified 2 events out of the 3 flagged events. Therefore, the precision of our approach is 66.7 %.

We empirically evaluate the improvement over our previous approach by using our previous approach to identify workload signature differences. Our previous approach identifies the following 4 workload signature differences:

```
FATAL org.apache.hadoop.mapred.Child: Error running child :
java.lang.OutOfMemoryError: Java heap space
   at org.apache.hadoop.io.Text.setCapacity(Text.java:240)
   at org.apache.hadoop.io.Text.append(Text.java:216)
   at org.apache.hadoop.util.LineReader.readLine
   (LineReader.java:159)
   at org.apache.hadoop.mapred.LineRecordReader.next
   (LineRecordReader.java:133)
   at org.apache.hadoop.mapred.LineRecordReader.next
   (LineRecordReader.java:38)
   at org.apache.hadoop.mapred.MapTask$TrackedRecordReader.moveToNext
   (MapTask.java:236)
   at org.apache.hadoop.mapred.MapTask$TrackedRecordReader.next
   (MapTask.java:216)
   at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:48)
   at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:436)
   at org.apache.hadoop.mapred.MapTask.run(MapTask.java:372)
   at org.apache.hadoop.mapred.Child$4.run(Child.java:255)
   at java.security.AccessController.doPrivileged(Native Method)
   at javax.security.auth.Subject.doAs(Subject.java:415)
   at org.apache.hadoop.security.UserGroupInformation.doAs
   (UserGroupInformation.java:1121)
   at org.apache.hadoop.mapred.Child.main(Child.java:249)

INFO org.apache.hadoop.mapred.Task: Task attempt_id done

INFO org.apache.hadoop.mapred.MapTask: Starting flush of map output

INFO org.apache.hadoop.mapred.Task: Task:attempt_id is done. And is
in the process of commiting
```

Only the first workload signature difference describes a meaningful difference between the system's behaviour in the field and during the test. Therefore, the precision of our previous approach is 25 % (i.e., 1/4).

We also use the state-of-the-practice approach to identify execution events with the largest occurrence frequency difference between the field and the test. We find that the state-of-the-practice flags the same events as our approach. Therefore, the precision of the state-of-the-practice 66.7 % (i.e., 2/3). We also use a statistical comparison to identify the execution events that differ between the field and the test. A statistical comparison of the execution events flags 19 events. These events describe the lack of successful processing of all input files in the field compared to the test. For example, the `INFO org.apache.hadoop.mapred.Task: attempt_id is done. And is in the process of commiting` event occurs much more frequently in the test compared to the field. Therefore, the precision of the statistical comparison is 0 % because these events do not describe the most important differences between the field and the test (i.e., the events related to the `OutOfMemoryError` event).

4.1.3 The exotic songs application

The second Hadoop application used in this case study is the Exotic Songs application (Adam 2012). The Exotic Songs application was developed to leverage the Million Songs data set (Million Song Dataset 2012). The Million Songs data set contains meta-data for one million different songs (the data set is 236 GB). The data set was developed (1) to encourage research on scalable algorithms and (2) to provide a benchmark data set for evaluating algorithms (Million Song Dataset 2011). The Exotic Songs application analyzes the Million Song data set to find "exotic" (i.e., popular songs produced by artists that live far away from other artists) songs.

4.1.3.1 Compression enabled in the field We monitored the performance of the Hadoop Exotic Songs application during a performance test. The performance test workload consisted of the full Millions Songs data set. We followed the following Microsoft TechNet blog to deploy the underlying Hadoop cluster (Klose 2014). The cluster contains (1) one DNS server, (2) one master node and (3) ten worker nodes.

We are grateful to Microsoft for (1) providing us access to such a large-scale deployment and (2) working closely with us to setup and troubleshoot our deployment.

We then monitored the Hadoop Exotic Songs application in the field and found that the throughput (completed attempts/s) was much lower than the throughput achieved during testing. We also found that the CPU usage was much higher in the field compared to our performance test. Therefore, we compare the execution logs from the test and the field to determine whether our tests accurately represent the current conditions in the field.

Our approach identifies the following two workload signature differences:

```
INFO com.hadoop.compression.lzo.GPLNativeCodeLoader: Loaded
native gpl library

INFO com.hadoop.compression.lzo.LzoCodec: Successfully loaded
& initialized native-lzo library
```

These execution events indicate that the Exotic Songs application is loading the Hadoop LZO compression libraries in the field. LZO is a fast and lossless data compression algorithm that is widely used in the field. The Hadoop LZO compression libraries support (1) splitting LZO files for distributed processing and (2) (de)compressing streaming data (input and output streams) (Hadoop-LZO 2011). Made aware of this issue, performance analysts could configure compression during performance testing to better understand the performance of their system. Hence, we have correctly identified 2 events out of the 2 flagged events. Therefore, the precision of our approach is 100 %.

We empirically evaluate the improvement over our previous approach by using our previous approach to identify workload signature differences. However, no execution events were flagged using our previous approach.

We also use (1) the state-of-the-practice approach and (2) a statistical comparison to identify execution events with the largest occurrence frequency difference between the field and the test. We find that these approaches both flag the same events as our approach. Therefore, the precision of these approaches is 100 % (i.e., 2/2).

4.2 Enterprise system case study

Although our Hadoop case study was promising, we perform three case studies on an enterprise system to examine the scalability of our approach. We note that these data sets are much larger than our Hadoop data set (see Table 8).

4.2.1 The enterprise system

Our second system is a large-scale enterprise software system in the telecommunications domain. For confidentiality reasons, we cannot disclose the specific details of the system's architecture, however the system is responsible for simultaneously processing millions of client requests and has very high performance requirements.

Performance analysts perform continuous performance testing to ensure that the system continuously meets its performance requirements. Therefore, analysts must continuously ensure that the performance tests accurately represent the current conditions in the field.

4.2.2 Comparing use-case performance tests to the field

Our first enterprise case study describes how our approach was used to validate a use-case performance test (i.e., a performance test driven by a workload generator) by comparing the system behaviour during the test and in the field. A workload generator was configured to simulate the individual behaviour of thousands of users by concurrently sending requests to the system based on preset use-cases. The system had recently added several new clients. To ensure that the existing use-cases accurately represent the workloads driven by these new clients, we use our approach to compare a test to the field.

We use our approach to generate workload signatures for each user within the test and in the field. We also generate workload signatures for each 1, 3 and 5 min time interval. We then compare the workload signatures generated during the test to those generated in the field. Our approach identifies 28 execution events, that differ between the workload signatures of the test and the field. These results were then given to multiple, independent system experts who confirmed:

1. Twenty-four events are under-stressed in the test relative to the field. In general, these events relate to variations in the intensity (i.e., changes in the number of events per second) of events in the field compared to the relatively steady-state of the test.
2. Two events are over-stressed in the test relative to the field.
3. Two events are artifacts of the difference in configuration between the test and field environments (i.e., these events correspond to communication between the

system and an external system that only occurs in the field) and are not important differences between the test and the field.

In summary, our approach correctly identifies 26 execution events (92.9 % precision) that correspond to important differences between the system's behaviour during the test and in the field. Such results can be used to improve the tests in the future (i.e., by tuning the use-cases and the workload generator to more accurately reflect the field conditions).

In contrast, our previous approach has a precision of 80 %. However, only 4 workload signature differences were correctly identified (compared to the 26 workload signature differences correctly identified by our new approach). Further, the state-of-the-practice approach has a precision of only 42.9 % and a statistical comparison flags 201 events with a precision of only 14.9 %.

4.2.3 Comparing replay performance tests to the field

Our second enterprise case study describes how our approach was used to validate a performance replay test (i.e., a performance test driven by a replay script) by comparing the system behaviour across a test and the field.

Replay scripts record the behaviour of real users in the field then play back the recorded behaviour during a replay test, where heavy instrumentation of the system is feasible. In theory, replay scripts can be used to perfectly replicate the conditions in the field during a replay test (Krishnamurthy et al. 2006). However, replay scripts require complex software to concurrently simulate the millions of users and billions of requests captured in the field. Therefore, replay scripts do not scale well and use-case performance tests that are driven by workload generators are still the norm (Meira et al. 2012).

Performance analysts monitoring the system's behaviour in the field observed a spike in memory usage followed by a system crash. We use our approach to understand the cause of this crash, and why it was not discovered during testing. Our approach identifies 5 influential execution events that differ between the workload signatures of the replay test and the field.

These results were given to performance analysts who confirmed that these 5 events are under-stressed in the replay test relative to the field. In particular, these events cause errors when over-stressed by an individual user (i.e., one user executing the event 1,000 times has a different impact on the system's behaviour than 100 users each executing the event 10 times). This type of behaviour cannot be identified from the occurrence frequencies or aggregate event counts.

In summary, our approach correctly identifies 5 influential execution events that correspond to differences between the system's behaviour during the replay test and in the field. Using this information, performance analysts update their replay tests. They then see the same behaviour during testing as in the field. Therefore, our results provide performance analysts with a very concrete recommendation to help diagnose the cause of this crash.

In contrast, our previous approach flags 26 events with a precision of 26.9 %. Further, the state-of-the-practice approach has a precision of 0 % and a statistical comparison flags 5 events with a precision of 0 %.

4.2.4 Comparing field-representative performance tests and the field

Our third enterprise case study describes how our approach was used to validate a field-representative performance replay test (i.e., a performance test driven by a replay script) by comparing the system behaviour across a performance test and the field. This test is known to be field-representative because it was successfully used to replicate a performance issue (i.e., a memory leak) in the field.

We use our approach to validate whether this test is truly representative of the field. Our approach identifies that no execution events differ between the workload signatures of the replay test and the field. Therefore, our results provide performance analysts with confidence in this test.

As our approach did not identify any execution events that differ between the workload signatures of the replay test and the field, we cannot compare our approach against the state-of-the-practice (i.e., the state-of-the-practice would examine the top 0 events). However, our previous approach flags 74 events. This is because our previous approach assumes that workload signature differences exist and proceeds to identify the largest such differences (i.e., our previous approach does not ignore differences below some threshold value). In addition, a statistical comparison of the execution events flags 2 events. These two events are artifacts of the test (i.e., these events correspond to functionality used to setup the tests) and are not important differences between the test and the field. Therefore, the precision of a statistical comparison is 0 %.

Our approach flags events with an average precision of 90%, outperforming our previous approach, the state-of-the-practice approach and the statistical comparison approach.

5 Discussion

5.1 Comparison to other approaches

Our case studies have shown that our approach performs well (average precision of 90 %) when detecting differences between the execution logs describing the test and field workloads. In particular, our approach outperforms (1) our previous approach (average precision of 63.0 %), (2) the state-of-the-practice (average precision 53.6 %) and (3) a basic statistical comparison of the execution logs from a test and the field (average precision of 3.0 %).

One reason for this outperformance may be that our approach breaks the workloads into workload signatures that represent two complementary components of the workload (i.e., the individual and aggregated user behaviour). Our approach then clusters the workload signatures and detects outlying clusters. Finally, our approach uses *unpaired two-sample two-tailed t-tests* to detect workload signature differences and *Cohen's d* to to filter unimportant differences. However, we used the same statistical tests to compare the workloads without breaking them down into workload signatures. The precision of our approach (90 %) is considerably greater than these statistical tests alone (3 %).

5.2 Formulations of the workload signature

Our approach also uses two complimentary formulations of the workload signatures (i.e., workload signatures representing (1) the individual user behaviour and (2) the aggregated user behaviour). These two formulations are able to detect different types of workload differences.

Workload signatures that represent the aggregated user behaviour are able to identify a set of execution events that occur together in the field, but not in the test. In our two Hadoop case studies, all of the execution events that were flagged were identified as signature differences between the workload signatures that represent the aggregated user behaviour. (i.e., the workload signatures representing the individual user behaviour were not able to detect workload differences between the field and the test). This is not surprising because the cause of the workload differences (i.e., failure of the Hadoop HDFS datanode in the field and no line-feeds or carriage-returns in any of the input files in the field) affect all users (i.e., the attempts). Similarly, in our first enterprise case study, all of the execution events that were flagged were identified as signature differences between the workload signatures that represent the aggregated user behaviour. This is also not surprising because most of these differences relate to variations in the intensity (i.e., changes in the number of events per second) of events in the field compared to the relatively steady-state of the test.

Workload signatures that represent the individual user behaviour are able to identify users whose behaviour is seen in the field, but not in the test. In our second enterprise case study, all of the execution events that were flagged were identified as signature differences between the workload signatures that represent the individual user behaviour. This is not surprising because these differences relate to an event that cause errors when over-stressed by an individual user.

5.3 Sensitivity analysis

The clustering phase of our approach relies on three different statistical measures: (1) a distance measure (to determine the distance between each workload signature), (2) a linkage criterion (to determine which clusters should be merged during the hierarchical clustering procedure) and (3) a stopping rule (to determine the number of clusters by cutting the hierarchical cluster dendrogram). We verify that these measures perform well when clustering workload signatures. Therefore, we determine the distance measure, linkage criterion and stopping rule that give our approach the highest precision using our Hadoop case study data (similar results hold for our other case studies). This analysis also serves to analyze the sensitivity of our results to changes in these measures.

Table 9 Determining the distance measure

Distance measure	#Events	Precision (%)
Pearson distance	12	91.7
Cosine distance	6	33.3
Euclidean distance	29	72.4
Jaccard distance	26	76.9
Kullback–Leibler divergence	33	72.7

Table 10 Determining the linkage criteria

Distance measure	#Events	Precision (%)
Average	12	91.7
Single	24	79.2
Ward	14	85.7
Complete	0	NA

5.3.1 Determining the distance measure

The agglomerative hierarchical clustering procedure begins with each performance signature in its own cluster and proceeds to identify and merge clusters that are "close." The "closeness" of two clusters is measured by some distance measure. The best known distance measure will result in a clustering that is closest to the manually assigned clusters.

We determine the distance measure by comparing the results obtained by our approach (i.e., the execution events that we flag) when different distance measures are used. Table 9 presents how the number of flagged events and the precision (the percentage of correctly flagged events) is impacted by several common distance measures (Fulekar 2008; Cha 2007; Frades and Matthiesen 2009). From Table 9, we find that the Pearson distance produces results with higher precision than any other distance measure.

5.3.2 Determining the linkage criteria

The hierarchical clustering procedure takes a distance matrix and produces a dendrogram (i.e., a hierarchy of clusters). The abstraction from a distance matrix to a dendrogram results is some loss of information (i.e., the distance matrix contains the distance between each pair of workload signatures, whereas the dendrogram presents the distance between each cluster). The best known linkage criterion will enable the hierarchical clustering procedure to produce a dendrogram with minimal information loss.

We determine the linkage criterion by comparing the results obtained by our approach (i.e., the execution events that we flag) when different distance measures are used. Similar to our analysis of the distance measure, Table 10 presents how the number of flagged events and the precision is impacted by several common linkage criteria (Frades and Matthiesen 2009; Tan et al. 2005). From Table 10 we find that the average linkage criterion produces results with higher precision than any other linkage criteria.

Table 11 Determining the stopping rule

Distance measure	#Events	Precision (%)
Calinski–Harabasz	12	91.7
Duda and Hart	0	NA
C-Index	31	71.0
Gamma	29	69.0
Beale	2	50
Cubic clustering criterion	0	NA
Point–Biserial	13	92.3
G(+)	29	69.0
Davies and Bouldin	27	74.1
Stepsize	0	NA

5.3.3 Determining the stopping rule

To complete the clustering procedure, dendrograms must be cut at some height so that each workload signature is assigned to only one cluster. Too few clusters will not allow outliers to emerge (i.e., they will remain nested in larger clusters) while too many clusters will lead to over-fitting and many false positives.

We determine the stopping rule by comparing the results obtained by our approach (i.e., the execution events that we flag) when different stopping rules are used. Similar to our analysis of the distance measure, Table 11 presents how the number of flagged events and the precision is impacted by several common linkage criteria (Milligan and Cooper 1985).

From Table 11, we find that the Calinski–Harabasz and Point–Biserial stopping rules have the greatest precision, 92.3 and 91.7 % respectively. We select the Calinski–Harabasz stopping rule because it is has a slightly higher precision than the Point–Biserial stopping rule across our other case studies. The Calinski–Harabasz stopping rule is also widely accepted as the best known stopping rule and used as the benchmark stopping rule.

6 Threats to validity

6.1 Threats to construct validity

6.1.1 The sequencing of events

Our approach does not explicitly consider the sequencing of events (although some of this information is reflected in the workload signatures that represent the aggregate user behaviour). Therefore, our approach may not be able to detect differences in the sequencing of events during the test compared to the field. However, event sequencing requires reliable information regarding the timing of events within the system and timing information may not be reliable in large-scale distributed systems (Lamport 1978).

6.1.2 Statistical tests

Our approach relies on two statistical tests. First, we use a *one-sample upper-tailed z-test for a population proportion* to identify outlying clusters. We then use an *unpaired two-sample two-tailed Welch's unequal variances t-test* (along with a *Cohen's d* effect size) to identify workload signature differences. These tests were able to identify meaningful differences between the performance test and the field in our case studies. However, these tests have assumptions that should be met for their proper application. Failure to satisfy these assumptions may affect the precision of our approach and undermine the statistical robustness of our approach.

The *one-sample upper-tailed z-test for a population proportion* and the *unpaired two-sample two-tailed Welch's unequal variances t-test* assume that:

1. The sampling is independent and random – this assumption is satisfied because our approach considers (1) all of the execution logs to generate workload signatures, (2) all of the clusters to identify outlying clusters and (3) all of the unique execution events to identify signature differences (i.e., selecting one execution log, cluster or unique execution event has no influence on the selection of another execution log, cluster or unique execution event). Further, we assume that the test logs are independently and randomly sampled from the test (i.e., the test can replicated). We also assume that the field logs are independently and randomly sampled from the field (see Sect. 6.2.1).
2. The population is normally distributed—this assumption is satisfied by the central limit theorem (Elliott 2006).

The *Welch's unequal variances t-test*, unlike the *Student's t-test*, does not assume that the sample sizes or the population variances are equal.

In addition to the underlying test assumptions, our use of statistical tests poses a second threat to validity. These tests may fail to correctly identify the most meaningful differences between a performance test and the field. For example, our thresholds (i.e., $p < 0.01$ and $d > 1.4$) may be too lenient (raising our recall, but lowering our precision) or too stringent (raising our precision, but lowering our recall). In the future, we intend to evaluate the impact of these thresholds on the precision and relative recall of our approach.

6.1.3 Evaluation

We have evaluated our approach by determining the precision with which our approach flags execution events that differ between the workload signatures of a performance test and the field. While performance analysts have verified these results, we do not have a gold standard data set. Further, complete system knowledge would be required to exhaustively enumerate every difference between a particular test and the field. Therefore, we cannot calculate the recall of our approach. However, our approach is intended to help performance analysts identify differences between a test and the field by flagging execution events for further analysis (i.e., to provide performance analysts with a starting point). Therefore, our goal is to maximize precision so that analysts have confidence in our approach. In our experience working with industry experts, performance analysts agree with this view (Hassan and Flora 2007; Jiang 2013; Syer et al. 2011b, 2013, 2014). Additionally, we were able to identify at least one execution event that differed between the test and the field in all of our case studies (except our enterprise case study where no differences were expected). Hence we were able to evaluate the precision of our approach in our case studies.

6.2 Threats to internal validity

6.2.1 Field-representative field logs

Our approach determines whether a performance test is field-representative by comparing execution logs from the test and the field. Therefore, our approach assumes that the execution logs from the field are field-representative. However, these execution logs describe the field workload over a specific period of time (e.g., a few hours or a few days). Field workloads may have short-term workload anomalies that are not considered field-representative. For example, during a 2013 movie premiere, Twitter's workload spiked to 143,199 tweets per second compared to an average of 5700 tweets per second (Twitter 2013). Such short-term workload anomalies may lead performance analysts to compare their performance test against a field workload that is not representative of the expected field workloads.

This threat can be mitigated by performance analysts. Performance analysts should use execution logs that describe the field workload over a period of normal operation. Performance analysts may also use execution logs that describe the field workload over longer periods of time (i.e., where short-term workload anomalies are less influential) because our approach has been designed to scale. However, our approach can be used to compare performance test workloads to the field despite short-term workload anomalies. Such comparisons may help performance analysts understand and replicate field issues (e.g., our Hadoop case studies and our second enterprise case study).

6.2.2 Execution log quality/coverage

Our approach generates workload signatures by characterizing a user's behaviour in terms of feature usage expressed by the execution events. However, it is possible that there are no execution logs to indicate when certain features are used. Therefore, our approach is incapable of identifying these features in the event that their usage differs between a test and the field. However, this is true for all execution log based analysis, including manual analysis.

This threat may be mitigated by using automated instrumentation tools that would negate the need for developers to manually insert output statements into the source code. However, we leave this to future work as automated instrumentation imposes a heavy overhead on the system and is often unfeasible in the field (Cornelissen et al. 2009; Uh et al. 2006; Bernat and Miller 2011; Laurenzano et al. 2015). Further, Shang et al. report that execution logs are a rich source of information that are used by developers to convey important information about a system's behaviour (Shang et al.

2011). Hence, automated instrumentation tools may not provide as deep an insight into the system's behaviour as execution logs.

6.2.3 Defining users for signature generation

In our experience, large-scale systems are typically driven by human agents. However, users may be difficult to define in systems that are driven by software agents (e.g., web services (Greenwood et al. 2007)) or when users are allowed to have multiple IDs. Defining the users of a particular system is a task for the system experts. However, such a determination only needs to be made the first time our approach is used, afterwards this may be definition is reused.

6.2.4 Defining the aggregated users for signature generation

Our approach generates workload signatures that represent the aggregated users are generated by grouping the execution logs into time intervals (i.e., grouping the execution logs that occur between two points in time). In our case studies, we used multiple time intervals (i.e., 1, 3 and 5 min time intervals) to generate these workload signatures. However, these time intervals may not produce the optimal results (e.g., using a 90 s time interval may have resulted in higher precision). We have mitigated this threat by using multiple time intervals that are known to generate accurate workload signatures for the particular systems in our case studies (Syer et al. 2013). However, it is possible that these time intervals are specific to the systems in our case studies.

6.3 Threats to external validity

6.3.1 Generalizing our results

The studied software systems represent a small subset of the total number of large-scale software systems. Therefore, it is unclear how our results will generalize to additional software systems, particularly systems from other domains (e.g., e-commerce). However, our approach does not assume any particular architectural details. Hence, there is no barrier to our approach being applied to other systems. Further, we have evaluated our approach on two different systems: (1) an open-source distributed data processing system and (2) an enterprise telecommunications system that is widely used in practice.

The clustering phase of our approach relies on three different statistical measures: (1) a distance measure (to determine the distance between each workload signature), (2) a linkage criterion (to determine which clusters should be merged during the hierarchical clustering procedure) and (3) a stopping rule (to determine the number of clusters by cutting the hierarchical cluster dendrogram). We evaluated several possible distance measures, linkage criterion and stopping rules by determining which ones gave our approach the highest precision. While the same set of statistical measures performed well across our five case studies, these measures may not generalize to other software systems. However, the main contribution of our work is the overall approach to comparing performance tests to the field, rather than determining a universal distance measure, linkage criteria and stopping rule for clustering workload signatures.

Our approach may not perform well on small data sets (where we cannot generate many workload signatures) or data sets where one set of execution logs (either the test or field logs) is much larger than the other. However, the statistical measures that we have chosen are invariant to scale. Further, we have evaluated our approach on small data sets. In our first Hadoop WordCount case study, the logs from the performance test only have 3,862 execution events (millions or billions of events are expected in large Hadoop deployments (Chen et al. 2012)). We have also evaluated our approach on a data set where one set of execution logs is much larger, on a relative and/or absolute basis, than the other. In our second Hadoop WordCount case study, the field logs are 6.6 times larger than the test logs and in our first enterprise case study, the field logs contain 2.5 million more execution events than the test logs.

7 Related work

This paper presented an automated approach to validate performance tests by comparing execution logs from a test and the field. Performance test design and log analysis are the most closely related areas of research to our work.

7.1 Performance test design

Much of the work in performance testing has focused on the automatic generation of tests (Avritzer and Weyuker 1995, 1994; Zhang and Cheung 2002; Draheim et al. 2006; Cai et al. 2007). A survey of performance testing (and performance test design) may be found in (Jiang 2013). Our approach may be used to validate performance tests by comparing these tests to the field. We intend to explore how the results of our approach may be used to automatically update performance tests.

7.2 Log analysis

Log analysis has received much attention in recent years (Shang 2014). In particular, workload characterization using execution logs is related to our work (Barros et al. 2007; Menascé 2002; Hassan et al. 2008; Nagappan et al. 2009; Kavulya et al. 2010; Draheim et al. 2006; Cai et al. 2007). However, such work has not been extended to compare performance test workloads to the field.

Shang et al. flag deviations in execution sequences mined from the execution logs of a test deployment and a field deployment of a large-scale system (Shang et al. 2013). Their approach reports deviations with a comparable precision to traditional keyword search approaches (23 % precision), but reduces the number of false positives by 94 %. Our approach does not rely on mining execution sequences. We also do not require any information regarding the timing of events within the system, which may be unreliable in distributed systems (Lamport 1978).

Jiang et al. flag performance issues in specific usage scenarios by comparing the distribution of response times for the scenario against a baseline derived from previous tests (Jiang et al. 2009). Their approach reports scenarios that have performance problems with few false positives (77 % precision). To overcome the need for a baseline, Jiang et al. mine execution logs to determine the dominant (expected) behaviour of the system and flag anomalies from the dominant behaviour (Jiang et al. 2008b). Their approach is able to flag <0.01 % of the execution log lines for closer analysis by system experts. Our approach is interested in highlighting the differences between a test and the field, as opposed to just anomalous behaviour. However, our approach can identify anomalous behaviour if such behaviour occurs primarily in the field (i.e., our third Enterprise case study).

In our previous work, we proposed an approach to identify performance deviations in thread pools using performance counters (Syer et al. 2011a, b). This approach is able to identify performance deviations (e.g., memory leaks) with high precision and recall. However, this approach did not make use of execution logs. Therefore, we could not identify the underlying cause of these performance deviations. The approach presented in this paper is concerned with highlighting the differences between a test and the field, as opposed to just performance issues.

This work builds on our previous research comparing the behaviour of a system's users, in terms of feature usage expressed by the execution events, between a performance test and the field (Syer et al. 2014). We have added robust statistical tests to detect outlying clusters (i.e., we use a *one-sample upper-tailed z-test for a population proportion* to detect outlying clusteres). We have also added robust statistical tests to detect workload signature differences (i.e., we use an *unpaired two-sample two-tailed t-test* to detect workload signature differences and *Cohen's d* to to filter unimportant differences).

We have also added a new type of workload signature (i.e., the workload signatures representing the aggregated user behaviour). This workload signature is partly based on the time-slice profiles introduced in our previous work (Syer et al. 2013). A time-slice profile is generated for each time-slice (i.e., the time between two successive samples of the performance counters) by (1) counting the number of times that each type of execution event occurred during the time-slice and (2) calculating the change in resource usage between the start and end of the time-slice. However, unlike a time-slice profile, a workload signature does not contain any information about the performance counters. Further, unlike our previous work (Syer et al. 2013), this paper compares the workload signatures from a performance test to the workload signatures from the field, rather than comparing the time-slice profiles of a single performance test to diagnose memory-related issues.

8 Conclusions and future work

This paper presents an automated approach to validate performance tests by comparing workload signatures from tests and the field using execution logs. Such signatures characterize user behaviour in terms of feature usage expressed in the execution logs. Our approach identifies differences between a test and the field. Such differences may be used by performance analysts to update their tests to more accurately reflect the field workloads.

We performed six case studies on two systems: one open-source system and one enterprise system. Our case studies explored how our approach can be used to identify feature differences, intensity differences and issue differences between performance tests and the field. Performance analysts and system experts have confirmed that our approach provides valuable insights that help to validate their tests and to support the continuous performance testing process.

Although our approach performed well, we intend to explore how well our approach performs when comparing additional data sets as well as data sets from other systems. We also intend to assess whether updating a performance test based on our approach results in the system's performance during the test becoming more aligned with the system's performance in the field. Finally, we intend to extend our approach with an interactive browser such that performance analysts can examine the results in greater detail and modify various aspects of our approach (e.g., how outlying clusters or signature differences are detected).

Acknowledgements We would like to thank BlackBerry for providing access to the enterprise system used in our case study. The findings and opinions expressed in this paper are those of the authors and do not necessarily represent or reflect those of BlackBerry and/or its subsidiaries and affiliates. Moreover, our results do not reflect the quality of BlackBerry's products. We would also like to thank Microsoft Azure for (1) providing us access to a large-scale deployment and (2) working closely with us to setup and troubleshoot our deployment.

References

Adam K.: Process a million songs with apache pig. http://blog.cloudera.com/blog/2012/08/process-a-million-songs-with-apache-pig/ (2012). Accessed 28 Oct 2015

Ausick, P.: NASDAQ gets off cheap in Facebook IPO SNAFU. http://finance.yahoo.com/news/nasdaq-gets-off-cheap-facebook-174557126.html (2012). Accessed 09 Dec 2014

Avritzer, A., Weyuker, E.J.: Generating test suites for software load testing. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 44–57 (1994)

Avritzer, A., Weyuker, E.J.: The automatic generation of load test suites and the assessment of the resulting software. Trans. Softw. Eng. **21**(9), 705–716 (1995)

Barros, M.D., Shiau, J., Shang, C., Gidewall, K., Shi, H., Forsmann, J.: Web services wind tunnel: on performance testing large-scale stateful web services. In: International Conference on Dependable Systems and Networks, pp. 612–617 (2007)

Bataille, J.: Operational progress report. http://www.hhs.gov/digitalstrategy/blog/2013/12/operational-progress-report.html (2013). Accessed 01 Jun 2014

Benoit, D.: Nasdaqs blow-by-blow on what happened to Facebook. http://blogs.wsj.com/deals/2012/05/21/nasdaqs-blow-by-blow-on-what-happened-to-facebook/ (2013). Accessed 05 May 2014

Bernat, A.R., Miller B.P.: Anywhere, any-time binary instrumentation. In: Proceedings of the Workshop on Program Analysis for Software Tools, pp. 9–16 (2011)

Bertolotti, L., Calzarossa, M.C.: Models of mail server workloads. Perform. Eval. **46**(2–3), 65–76 (2001)

Cai, Y., Grundy, J., Hosking, J.: Synthesizing client load models for performance engineering via web crawling. In: Proceedings of the International Conference on Automated Software Engineering, pp. 353–362 (2007)

Calinski, T., Harabasz, J.: A dendrite method for cluster analysis. Commun. Stat. **3**(1), 1–27 (1974)

Cha, S.H.: Comprehensive survey on distance/similarity measures between probability density functions. Int. J Math. Models Methods Appl. Sci. **1**(4), 300–307 (2007)

Chen, Y., Alspaugh, S., Katz, R.: Interactive analytical processing in big data systems: a cross-industry study of mapreduce workloads. Proc. VLDB Endow. **5**(12), 1802–1813 (2012)

Cheng, J.: Steve jobs on MobileMe. http://arstechnica.com/apple/2008/08/steve-jobs-on-mobileme-the-full-e-mail/ (2008). Accessed 25 Jan 2014

Cohen, J.: Statistical Power Analysis for the Behavioral Sciences, 2nd edn. Routledge, New York (1988)

Coleman P.: The avoidable cost of downtime. http://www.ca.com//media/Files/SupportingPieces/acd_report_110110.ashx (2011). Accessed 14 Apr 2014

Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L., Koschke, R.: A systematic survey of program comprehension through dynamic analysis. Trans. Softw. Eng. **35**(5), 684–702 (2009)

Dean, J., Barroso, L.A.: The tail at scale. Commun. ACM **56**(2), 74–80 (2013)

Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)

Draheim, D., Grundy, J., Hosking, J., Lutteroth, C., Weber, G.: Realistic load testing of web applications. In: Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 57–68 (2006)

Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis, 1st edn. Wiley, New York (1973)

Elliott, A.C.: Statistical Analysis Quick Reference Guidebook, 1st edn. Sage, Thousand Oaks (2006)

Frades, I., Matthiesen, R.: Overview on techniques in cluster analysis. Bioinform. Methods Clin. Res. **593**, 81–107 (2009)

Fulekar, M.H.: Bioinformatics: Applications in Life and Environmental Sciences, 1st edn. Springer, New York (2008)

Greenwood, D., Lyell, M., Mallya, A., Suguri, H.: The IEEE FIPA approach to integrating software agents and web services. In: Proceedings of the International Joint Conference on Autonomous-Agents and Multiagent Systems, pp. 1412–1418 (2007)

Hadoop: http://hadoop.apache.org/ (2014). Accessed 17 Apr 2013

Hadoop-LZO: https://github.com/twitter/hadoop-lzo (2011). Accessed 28 Oct 2015

Harris, C.: IT downtime costs $26.5 billion in lost revenue. http://www.informationweek.com/it-downtime-costs-$265-billion-in-lost-revenue/d/d-id/1097919? (2011). Accessed 25 Jan 2014

Hassan, A.E., Flora, P.: Performance engineering in industry: current practices and adoption challenges. In: Proceedings of the International Workshop on Software and Performance, pp. 209–209 (2007)

Hassan, A.E., Martin, D.J., Flora, P., Mansfield, P., Dietz, D.: An industrial case study of customizing operational profiles using log compression. In: Proceedings of the 30th International Conference on Software Engineering, pp. 713–723 (2008)

Howell Jr., T., Dinan, S.: Price of fixing, upgrading obamacare website rises to $121 million. http://www.washingtontimes.com/news/2014/apr/29/obamacare-website-fix-will-cost-feds-121-million/ (2014). Accessed 09 Dec 2014

Huang, A.: Similarity measures for text document clustering. In: Proceedings of the New Zealand Computer Science Research Student Conference, pp. 44–56 (2008)

Jiang Z.M.: Automated analysis of load testing results. PhD thesis, Queen's University (2013)

Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P.: An automated approach for abstracting execution logs to execution events. J. Softw. Maint. Evol. **20**(4), 249–267 (2008a)

Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P.: Automatic identification of load testing problems. In: Proceedings of the International Conference on Software Maintenance, pp. 307–316 (2008b)

Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P.: Automated performance analysis of load tests. In: Proceedings of the International Conference on Software Maintenance, pp. 125–134 (2009)

Kampenes, V.B., Dybå, T., Hannay, J.E., Sjøberg, D.I.K.: A systematic review of effect size in software engineering experiments. Inform. Softw. Technol. **49**(11–12), 1073–1086 (2007)

Kavulya, S., Tan, J., Gandhi, R., Narasimhan, P.: An analysis of traces from a production mapreduce cluster. In: Proceedings of the International Conference on Cluster, Cloud and Grid Computing, pp. 94–103 (2010)

Klose, O.: Hadoop on Linux on Azure. http://blogs.technet.com/b/oliviaklose/archive/2014/06/17/hadoop-on-linux-on-azure-1.aspx (2014). Accessed 28 Oct 2015

Kremenek, T., Engler, D.: Z-ranking: using statistical analysis to counter the impact of static analysis approximations. In: Proceedings of the International Conference on Static Analysis, pp. 295–315 (2003)

Krishnamurthy, D., Rolia, J.A., Majumdar, S.: A synthetic workload generation technique for stress testing session-based systems. Trans. Softw. Eng. **32**(11), 868–882 (2006)

Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)

Laurenzano, M.A., Peraza, J., Carrington, L., Tiwari Jr., A., Ward, W., Campbell, R.: Pebil: binary instrumentation for practical data-intensive program analysis. Clust. Comput. **1**(18), 1–14 (2015)

MapReduce Tutorial: http://hadoop.apache.org/docs/stable/mapred_tutorial.html (2014). Accessed 16 Jun 2014

Meira, J.A., de Almeida, E.C., Traon, Y.L., Sunye, G.: Peer-to-peer load testing. In: Proceedings of the International Conference on Software Testing, Verification and Validation, pp. 642–647 (2012)

Menascé, D.A.: Load testing of web sites. IEEE Internet Comput. **6**(4), 70–74 (2002)

Metrics 20: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/metrics2/package-summary.html (2014). Accessed 16 Jun 2014

Milligan, G.W., Cooper, M.C.: An examination of procedures for determining the number of clusters in a data set. Psychometrika **50**(2), 159–179 (1985)

Million Song Dataset: https://aws.amazon.com/datasets/million-song-dataset/ (2011). Accessed 28 Oct 2015

Million Song Dataset: http://labrosa.ee.columbia.edu/millionsong/ (2012). Accessed 28 Oct 2015

Mojena, R.: Hierarchical grouping methods and stopping rules: an evaluation. Comput. J. **20**(4), 353–363 (1977)

Nagappan, M., Wu, K., Vouk M.A.: Efficiently extracting operational profiles from execution logs using suffix arrays. In: Proceedings of the International Symposium on Software Reliability Engineering, pp. 41–50 (2009)

OutputCommitter: http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapred/OutputCommitter.html (2014). Accessed 16 Jun 2014

Parnas, D.L.: Software aging. In: Proceedings of the International Conference on Software Engineering, pp. 279–287 (1994)

PerfMon: http://perfmon.sourceforge.net/ (2014). Accessed 26 Jan 2014

RecordReader: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/RecordReader.html (2014). Accessed 16 Jun 2014

Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**(1), 53–65 (1987)

Sandhya, N., Govardhan, A.: Analysis of similarity measures with wordnet based text document clustering. In: Proceedings of the International Conference on Information Systems Design and Intelligent Applications, pp. 703–714 (2012)

Shang, W.: Log engineering: towards systematic log mining to support the development of ultra-large scale systems. PhD thesis, Queen's University (2014)

Shang, W., Jiang, Z.M., Adams, B., Hassan, A.E., Godfrey, M.W., Nasser, M., Flora, P.: An exploratory study of the evolution of communicated information about the execution of large software systems. In: Proceedings of the Working Conference on Reverse Engineering, pp. 335–344 (2011)

Shang, W., Jiang, Z.M., Hemmati, H., Adams, B., Hassan, A.E., Martin, P.: Assisting developers of big data analytics applications when deploying on hadoop clouds. In: Proceedings of the International Conference on Software Engineering, pp. 402–411 (2013)

Shang, W., Nagappan, M., Hassan, A.E.: Studying the relationship between logging characteristics and the code quality of platform software. Empir. Softw. Eng. 20(1), 20:1–20:27 (2015)

SiliconBeat: Firefox download stunt sets record for quickest meltdown. http://www.siliconbeat.com/2008/06/17/firefox-download-stunt-sets-record-for-quickest-meltdown/ (2008). Accessed 25 Jan 2014

Software Engineering Institute: Ultra-Large-Scale Systems: The Software Challenge of the Future. Carnegie Mellon University, Pittsburgh (2006)

Sokal, R.R., Rohlf, F.J.: Biometry: The Principles and Practice of Statistics in Biological Research, 4th edn. W. H. Freeman, New York (2011)

Student: The probable error of a mean. Biometrika 6(1), 1–25 (1908)

Syer, M.D., Adams, B., Hassan A.E.: Identifying performance deviations in thread pools. In: Proceedings of the International Conference on Software Maintenance, pp. 83–92 (2011a)

Syer, M.D., Adams, B., Hassan A.E.: Industrial case study on supporting the comprehension of system behaviour. In: Proceedings of the International Conference on Program Comprehension, pp. 215–216 (2011b)

Syer, M.D., Jiang, Z.M., Nagappan, M., Hassan, A.E., Nasser, M., Flora, P.: Leveraging performance counters and execution logs to diagnose memory-related performance issues. In: Proceedings of the International Conference on Software Maintenance, pp. 110–119 (2013)

Syer, M.D., Jiang, Z.M., Nagappan, M., Hassan, A.E., Nasser, M., Flora, P.: Continuous validation of load test suites. In: Proceedings of the International Conference on Performance Engineering, pp. 259–270 (2014)

Tan, P.N., Steinbach, M., Kumar, V.: Cluster Analysis: Basic Concepts and Algorithms, 1st edn. Addison-Wesley Longman Publishing Co., Inc, Boston (2005)

TextInputFormat: http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapred/TextInputFormat.html (2014). Accessed 16 Jun 2014

The Sarbanes-Oxley Act 2002: http://soxlaw.com/ (2014). Accessed 28 Jan 2014

Twitter: New Tweets per second record, and how! https://blog.twitter.com/2013/new-tweets-per-second-record-and-how (2013). Accessed 12 Dec 2014

Uh, G.R., Cohn, R., Yadavalli, B., Peri, R., Ayyagari, R.: Analyzing dynamic binary instrumentation overhead. In: Proceedings of the Workshop on Binary Instrumentation and Applications, pp. 56–64 (2006)

Voas, J.: Will the real operational profile please stand up? IEEE Softw. **17**(2), 87–89 (2000)

Welch, B.L.: The generalization of "student's" problem when several different population variances are involved. Biometrika **34**(1–2), 28–35 (1997)

Weyuker, E., Vokolos, F.: Experience with performance testing of software systems: issues, an approach, and case study. Trans. Softw. Eng. **26**(12), 1147–1156 (2000)

Williams, A.: Amazon web services outage caused by memory leak and failure in monitoring alarm. http://techcrunch.com/2012/10/27/amazon-web-services-outage-caused-by-memory-leak-and-failure-in-monitoring-alarm/ (2012). Accessed 09 Dec 2014

Yuan, D., Luo, Y., Zhuang, X., Rodrigues, G.R., Zhao, X., Zhang, Y., Jain, P.U., Stumm, M.: Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In: Proceedings of the Conference on Operating Systems Design and Implementation, pp. 249–265 (2014)

Zhang, J., Cheung, S.C.: Automated test case generation for the stress testing of multimedia systems. Softw. Pract. Exp. **32**, 1411–1435 (2002)

Zhang, Z., Cherkasova, L., Loo B.T. Benchmarking approach for designing a mapreduce performance model. In: Proceedings of the International Conference on Performance Engineering, pp. 253–258 (2013)