Reproducibility of environment-dependent software failures: An experience report

Davide G. Cavezza
Roberto Pietrantuono
Stefano Russo
Università di Napoli Federico II
80125, Napoli, Italy
d.cavezza@studenti.unina.it
roberto.pietrantuono@unina.it
sterusso@unina.it

Javier Alonso
Institute of Advanced Studies on Cybersecurity
University of Leon
Leon, Spain
javier.alonso@unileon.es

Kishor S. Trivedi
ECE Department
Duke University
27708, Durham, NC, USA
ktrivedi@duke.edu

***Abstract*—We investigate the dependence of software failure reproducibility on the environment in which the software is executed. The existence of such dependence is ascertained in literature, but so far it is not fully characterized.**

In this paper we pinpoint some of the environmental components that can affect the reproducibility of a failure and show this influence through an experimental campaign conducted on the MySQL Server software system. The set of failures of interest is drawn from MySQL's failure reports database and an experiment is designed for each of these failures.

The experiments expose the influence of disk usage and level of concurrency on MySQL failure reproducibility. Furthermore, the results show that high levels of usage of these factors increase the probabilities of failure reproducibility.

***Keywords*-Software testing, debugging.**

I. Introduction

Understanding the nature of software bugs and their manifestation is a daunting task in today's computing systems. Researchers have analyzed bugs from different viewpoints in order to improve the knowledge about their characteristics, and consequently improve software development processes. The need of comprehending the root cause of a bug led, in the past, to develop several schemes capturing static properties of software defects. Examples are the HP scheme, the IEEE 1044 classification, and the Orthogonal Defect Classification (ODC). Less commonly, researchers focus on the bug manifestation process, although it is increasingly complicated as systems grow in size and complexity. Expectedly, many bugs systematically cause the same failure on a given (sequence of) input(s). However, researchers have often observed that there is a non-negligible share of bugs that cause a failure depending on the state of the execution environment, and may appear even as non-deterministic or transient as the failure does not occur unless the environment is in a certain state. The latter category heavily hinders both testing and debugging activities, since their exposure and reproduction may be a rare event.

Some broad classifications, developed since the end of the eighties, take into account the properties of bugs related to the reproducibility of relevant failures. Gray's Bohrbug/Heisenbug classification [1], and its posterior refinement by Trivedi and Grottke considering Mandelbug in lieu of Heisenbug [2], distinguish bugs according to the complexity of their reproduction process. Despite their merit of pointing out this need for a better testing/debugging process, we are still far from gaining a deep insight into the failure reproduction process. The major difficulty is in characterizing and defining the failure reproduction "complexity" depending on environmental conditions.

The objective of this study is to investigate the impact of some environmental factors on the probability of reoccurrence of software failures. Considering the most common causes of environment-dependent failures, the factors taken into account are: memory occupancy, disk usage, and concurrency level.

The study is conducted through a controlled experiment in which we focus on 7 hard-to-reproduce failures of the Database Management System MySQL. The experiment consists in repeated executions of a determined workload under several controlled configurations of the environmental factors, to see how many times each of the failures surfaces in the different configurations.

Results highlight the systematic impact of some of the considered factors. Although these are still coarse-grained factors, this is a first step toward the characterization and control of the execution environment for improving reproducibility and supporting testing/debugging activities. The study output helps to formulate hypotheses about which factor of the environment is more likely to influence the failure reproduction process and worth further investigation in future research.

The rest of the paper is organized as follows. Section II summarizes the related work. Section III lists the research questions and introduces the case study. Section IV describes the failure selection criteria. Section V introduces the details of the experimental campaign conducted. Section VI presents the results of the experiments conducted and Section VII discusses the results obtained and highlights the threats to validity. Finally, Section VIII concludes the paper.

II. Related work

Several bug classifications have been proposed in literature, targeting different attributes of bugs. An overall view of the most popular classifications is provided in [3]. One of the most widespread is Orthogonal Defect Classification (ODC) [4]. It classifies bugs in terms of the kind of fix they underwent and the verification activity in which they were discovered. This classification has been successfully applied in different projects as a means to pinpoint and thereby eliminate the weaknesses in the processes that were mainly responsible for the discovered faults ([5], [6], [7], [8]).

Other popular schemes are HP's *Defect Origins, Types and Modes* [9] and IEEE's standard 1044 [10]. These schemes, like ODC, focus on static properties of bugs such as the software artifact affected by a bug and the verification activity in which it was found.

The issue of failure reproducibility is addressed by Gray in [1]. He classifies bugs in two categories: *Bohrbugs*, whose activation is easily reproduced by submitting a certain set of inputs to the system, and Heisenbugs, whose activation and propagation to the user interface are not deterministically reproducible. In [11], Grottke and Trivedi revise this classification by substituting the term Heisenbug with *Mandelbug*, since the former was originally used by Lindsay to indicate bugs that change their behavior when probed, actually a subtype of Mandelbug.

Mandelbugs constitute a non-negligible share of software faults even in critical systems. The authors in [2] perform a classification of faults reported in the flight software employed in 18 JPL/NASA space missions: 36.5% of the analyzed bugs were Mandelbugs. Similar or higher percentages are reported in the open-source programs analyzed in [12].

The environment in which software is executed plays a central role in failure reproduction. In [13] and [2] four factors of complexity in the failure reproduction process are pinpointed as responsible for a bug to be classified as Mandelbug: *i)* a time lag between the fault activation and the failure occurrence; *ii)* interactions of the software application with hardware, operating system or other applications running concurrently on the same system; *iii)* influence of the timing of inputs and operations; *iv)* influence of the sequencing of operations.

The last three factors are actually responsible for the randomness in failure reproduction. In [14] they are collectively referred as the *operating environment*. Consequently, failures are divided in two classes:

- *environment-independent*, which always occur in response to a given workload;
- *environment-dependent*, whose reoccurrence depends on the environment, in the sense that if the environment's state changes enough, the failures do not appear when the workload is resubmitted.

Further studies, despite using classification schemes different from the mentioned ones, highlight failures' characteristics attributable to environment-dependent ones. They are indirectly concerned with the issue of reproducibility, as their main focus is to investigate the effectiveness of fault tolerance methods based on retrying a failed task. The work in [15] analyzes 200 failure reports of the Tandem GUARDIAN system, a system previously studied by Gray in [1] and [16]; it reports examples of failures dependent on a particular state of memory. The study in [17] focuses on fault tolerance techniques employed in the earlier mentioned JPL/NASA space missions' software. The work in [18] models fault tolerance against Mandelbugs in 11 IT systems.

The mentioned works provide significant examples of the relationship between the environment and the failure reproduction process. However, although this relationship is explicitly stated, it has not been systematically characterized in the existing literature.

III. Research questions

The goal of our study is to investigate the reproducibility of environment-dependent failures (in the sense of [14]) and to characterize its dependence on the environment. The paper addresses the following questions:

- Can we characterize and isolate the environmental conditions that the failure reproduction depends on?
- Is it possible to set the factors in such a way as to improve failure reproducibility?

The answer to these questions is investigated with reference to the Database Management System MySQL Server. It is a good example of medium-scale system used also in critical contexts. The choice of MySQL is also due to the quality of its failure data. Most failure reports contain a *How to repeat* field in which a workload capable of reproducing the failure is specified. This field is a valuable source of information, as we are interested in reproducibility properties of failures: we use its content as a workload to exercise each failure in our experiments. We refer to it as the *failure-specific* workload.

The following sections describe our procedure for the selection of the failure reports, the design and the implementation of the experiments.

IV. Failures selection

MySQL has a very large amount of failure data, so we apply highly restrictive criteria to select which failures to include in our study. We focus on failures related to version 5.1 of MySQL Server. We restrict the research to *Closed* reports, so as to have complete information about the possible reproducing workload(s) and the fixes that were applied after each failure. With these criteria, 568 failure reports are returned by MySQL's failure database.

Through a manual inspection of the reports, we distinguish between environment-independent and environment-dependent failures; this analysis is based on: *i)* the presence of phrases indicating potentiality, like "sometimes fails", "it may fail"; *ii)* explicit reference to environmental factors, like disk or memory, in the description. We manually filter out all the environment-independent failures and concentrate our efforts on the 82 environment-dependent ones. On these we perform a second filtering to select the failures suitable to be tested in our experimental campaign. A summary of this second filtering is shown in Table I.

A first criterion failures must satisfy is detectability: it must be feasible to determine, from the inspection of the output, whether or not the failure appeared. As the easiest condition to detect is a crash of MySQL, we include failures of this kind; we also include failures exposed through MySQL Test Framework, which automatically detects wrong outputs in response to predefined test cases. Failures that do not fulfill both these requirements are labeled as *Not verifiable*: eighteen failures of this kind have been found and excluded from the study.

An example of this kind is the failure #35074. It consists in a wrong value of MySQL's status variable `max_used_connections`, which represents the maximum number of simultaneous connections opened since the server started. To detect the occurrence of such a failure we should be able to predict the correct value and compare it to the one reported by MySQL. Since in our experiment we open a random number of connections to implement the level of concurrency factor, we cannot know in advance the correct value. Therefore, we cannot detect this failure.

Some failures are not testable in our test bench, as their effects make it difficult to restore MySQL's initial state after every execution of the workload (see Section V-A). Let us consider for instance failure #55616: it requires the activation of MySQL's replication option; this in turn relies on the binary logging of operations; with binary logging, re-populating tables does not restore the initial state, as the deletion operations involved would be permanently logged. Three failures, labeled *Replication*, have thus been discarded.

Table I
FAILURE REPORTS' CLASSIFICATION SUMMARY

Label	Number of reports
Not verifiable	18
Replication	3
Error/Bug report	10
Workload not specified	17
Ambiguous workload	3
Version not available	12
Aging	8
Test case misconception	3
User mistake	1
Final considered reports	7
Total	82

Although most reports are actually *failure* reports, some of them notify internal errors or bugs of MySQL, not perceivable from the user interface. Ten failures, labeled *Error report* or *Bug report*, have been excluded. An example of error report is the #37044. It reports how to observe an erroneous value of an internal variable in the debugging tool `gdb`. This error does not necessarily lead to a failure, and in any case no workload that potentially cause a failure is provided. Therefore, it is impossible for us to include this report in our study.

A further requirement is that the failure-specific workload must be clearly specified and reproducible. As environment-dependent failures are difficult to reproduce, reporters are often unable to devise a workload in *How to repeat*; there are also cases in which workloads are attached as files that are no longer available (see failure #50227). Twenty failures have been labeled as *Workload not specified* or *Ambiguous workload*; they have been discarded from the study.

It is important to identify the exact subversion of MySQL in which the failure can be observed. There is a *Version* field for this purpose, but it is sometimes imprecise or useless: in some cases it contains only the version number 5.1, without any further specification about the subversion; other times it reports the identifiers of old branches in the SVN that are no longer available, or old releases that cannot be found on MySQL's website. To establish if a subversion is appropriate to test a failure, we verify one of the following criteria: *i)* by analyzing the patch referred in the failure report and comparing it to the subversion's source code, we notice that the patch has not been applied to the subversion; *ii)* we observe the failure at least once in a series of preliminary executions of the failure-specific workload. As regards this requirement, 12 failures have been classified as *Version not available* and excluded.

Some reports pertain memory leaks or other aging-related errors. This phenomenon has already been studied in [19] and is outside our scope. We have found and excluded 8 *Aging* reports. They concern memory leaks logged by the DBMS running inside a profiler, like *Valgrind*.

Three reports have been excluded because due to misconceptions in test cases rather than failures of MySQL. Moreover, the failure #13543 is not due to a bug as well, but to a user mistake. None of the testers was able to repeat the failure and the reporter finally declared that it was due to corrupted data files on his installation: he solved the problem by reinstalling MySQL and recreating his databases.

Finally, we narrow down our sample to 7 environment-dependent failures. Each one is tested on a specific subversion of MySQL, as shown in Table II. As of failures #32148 and #38691, although they were reported on MySQL 5.1, we have not found any subversion of the 5.1 that satisfied the criteria described earlier in this section; for this reason, we have decided to test them on subversion 5.0.67.

Table II
ASSOCIATIONS BETWEEN FAILURES AND MYSQL'S SUBVERSIONS

Failure	Subversion
32148	5.0.67
38691	5.0.67
38823	5.1.23a
42419	5.1.30
44521	5.1.34
46539	5.1.30
55421	5.1.34

V. EXPERIMENTAL METHOD

A. Design of experiment

For each of the selected failures we design an experiment to investigate the influence of a set of hypothesized factors on an outcome variable representative of the study's goals.

The first step of Design of Experiment (DoE) is the selection of the outcome variable. Our concern is to characterize reproducibility of software failures; we need to translate our goal in a quantifiable entity related to the system under consideration. A good candidate to represent reproducibility is the probability of reoccurrence of a failure over repeated submissions of the failure-specific workload; this quantity can be estimated by the proportion of observed failures over a fixed number of iterations of the workload. Therefore, our choice for the outcome variable is a binary variable that indicates *whether or not a failure occurred in a single execution of the failure-specific workload*; by counting the occurrences, we obtain the desired proportion. Relying on a similar study conducted on the browser Mozilla Firefox, described in [20], we fix the number of iterations to 10.

Next, we have to choose the factors to study. They should be characteristics of the environment that are likely to affect failures' reproducibility in the system. In our survey we are considering a DBMS. It is a system which makes large use of complex in-memory data structures in order to perform operations on tables, to keep the tables consistent, to cache them so as to get data in short time. On the other hand, it needs to access disk to memorize and retrieve data as long as relevant metadata. Finally, it is a concurrent system, which may have to handle several transactions at the same time, each one issued by a different user.

From these considerations, we assume that the factors in our experiment are:

- *memory occupation*, namely the percentage of memory occupied by MySQL's processes and threads over the total available memory;
- *disk usage*, that is the amount of bandwidth of transmission between disk and memory currently in use;
- *level of concurrency*, the number of users currently connected to MySQL and requesting operations to it.

These factors are also mentioned in other empirical studies ([14], [15], [21]) as examples of causes that may affect failure reproducibility.

We choose to adopt two levels, *high* and *low*, for each factor. This is common for exploratory studies, concerned more on exposing which factors are most likely to influence the outcome than on setting up a quantitative model [22].

The final step in the design is to define the treatments and in which order they have to be executed. We choose a *full-factorial* design, which tests each possible combination of levels for the factors. The order of treatments is completely randomized: this is done in order to avoid observing any variation of the outcome due to an effect of a chosen order, variation that may be wrongly attributed to the change in the factors' levels.

The final resulting plan is shown in Table III.

B. Architecture of the test bench

The testing environment consists of two virtual machines running on the same physical computer, an Acer Aspire with: CPU Intel Core i5-3230M, dual-core; 4 GB RAM; a 500 GB hard disk; Windows 8.1 operating system.

Each virtual machine is created and run by an instance of VMware Player. One of the two machines, referred as the *server machine*, is intended to run MySQL Server, while the other one, named *client machine*, hosts and executes the testing scripts and collects the results. The two VMs are connected via a LAN emulated by VMware as well.

The server virtual machine has the following resources: CPU single-core; 1 GB of primary memory; 40 GB of hard disk. The client virtual machine has: CPU single-core; 1 GB of primary memory; a 20 GB hard disk. The server machine is provided with more disk space as it needs to host several installations of MySQL Server along with the respective databases. On both machines we have installed Ubuntu 12.04.2 for Desktop.

C. Factors' implementation

We map memory occupation onto the percentage of physical memory assigned to the `mysqld` process by the OS; in Linux, physical memory assigned to a process is called *Resident Set*, and its amount *Resident Set Size* (*RSS*).

We choose physical rather than virtual memory because we think it is more suitable to represent a condition which may influence failure reproducibility. The reason of that lies in the particular way Linux handles virtual memory,

Table III
EXPERIMENTAL PLAN

ID	Memory occupation	Disk usage	Level of concurrency
1	High	Low	High
2	Low	Low	High
3	Low	High	High
4	Low	Low	Low
5	High	High	High
6	High	Low	Low
7	High	High	Low
8	Low	High	Low

called *overcommit* [23]: since many applications ask for more memory space than they actually use, the Linux kernel always responds positively to memory allocation requests (like C's `malloc`), even if the amount of requested space exceeds the physically available RAM, but the space is not allocated immediately; it will be allocated when the program actually accesses it for reading or writing. Thus, virtual memory is unbounded and it makes no sense to talk about percentage of allocated virtual memory, neither do we have any reason to believe that the program's behavior can be affected by the amount of requested virtual memory.

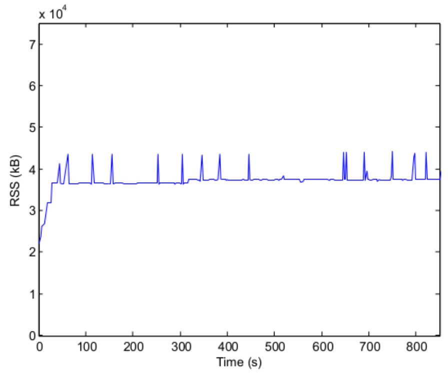
The RSS is the trickiest factor to control, since it depends on the OS's memory allocation policies. Several configuration options of MySQL may affect the RSS. We hypothesize that one of the most influential is the size of the memory buffer used by InnoDB, MySQL's Storage Engine, to cache data from tables [24]: it can be controlled via the configuration parameter `innodb_buffer_pool_size`.

Furthermore, in order to prevent MySQL from relying on the file system's write-back cache, we set the parameter `innodb_flush_method` to the value `O_DIRECT`. This is done because write-back cache would require kernel memory space outside `mysqld`'s addressing space and thereby would hinder memory monitoring.

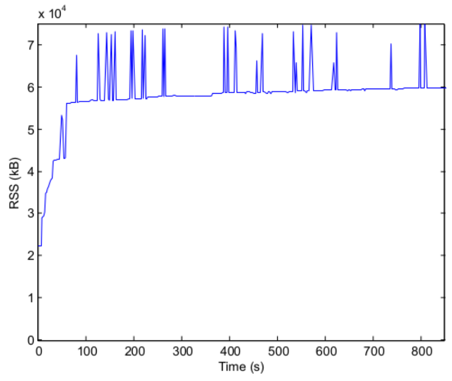
To have MySQL actually use the assigned memory, we employ the TPC-W benchmark [25] as a workload for it. In order to distinguish this workload from the failure-specific one, we will sometimes refer to it as the *conditioning workload*. We first profile its effect on physical memory allocation for different sizes of InnoDB's buffer. To monitor RSS during TPC-W execution, we use a script that every two seconds polls the `/proc/`pidof mysqld`/status` pseudofile for the *VmRSS* value.

Observations are plotted in Fig. 1. We can see that our hypothesis is confirmed: for a buffer of 8 MB, the RSS stabilizes around 38 MB, while for 60 MB it settles around 60 MB. We also notice that after 200 seconds of execution, both the configurations reach a stable value. Therefore, we decide to implement the two levels of memory occupation by submitting the conditioning workload for 200 seconds before submitting the failure-specific workload.

However, even in the high memory configuration, the reached value (60 MB) is too small if compared with the total memory available on the server machine (1 GB); we need a workaround to limit the maximum available memory for `mysqld`. *cgroups* (*control groups*) is a Linux kernel feature developed by Google's engineers [26]; among its functions there is resource limiting, which allows to set a maximum amount of memory allocatable to a process. Looking at the memory profiles, we choose to fix this limit to 70 MB. In this way, the amounts of RSS reached in the two configurations constitute two significantly different percentages of the available memory. This is shown in Table IV: the amount of occupied memory in the 8-MB-cache configuration is about half the total memory, while the one in the 60-MB-cache configuration is near the bound.



(a) Profile for a 8 MB buffer (in kB)



(b) Profile for a 60 MB buffer (in kB)

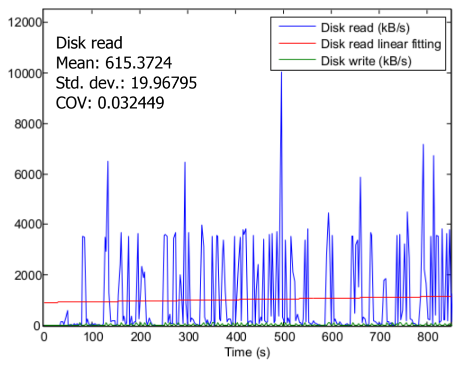
Figure 1. Memory occupation profiles for different values of `innodb_buffer_pool_size`

Disk usage is controlled by means of *PostMark*, a benchmark for Linux file system. It creates a bunch of files and performs iteratively reading and writing operations on them; at the end, it shows a report about the average and standard deviation of disk's reading and writing speeds.

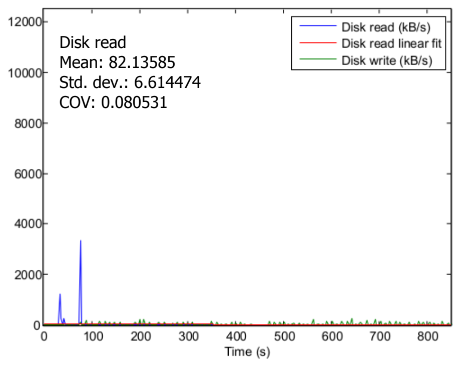
Fig. 2 and 3 show the disk usage of MySQL and PostMark over time. We choose to characterize it by its average value during the time interval set up for the conditioning of the environmental factors, namely 200 seconds. The figures indicate that the average is a good candidate to summarize the measured data: since the coefficients of variation (COV) are less than 0.1, we can assume that the mean is stable.

Table IV
MEMORY OCCUPATION LEVELS

Level	Steady state RSS	Percentage of available memory
Low	∼38 MB	∼54%
High	∼60 MB	∼85%



(a) Profile for an 8 MB buffer (in kB/s)



(b) Profile for a 60 MB buffer (in kB/s)

Figure 2. Disk usage profiles for different values of the parameter `innodb_buffer_pool_size`

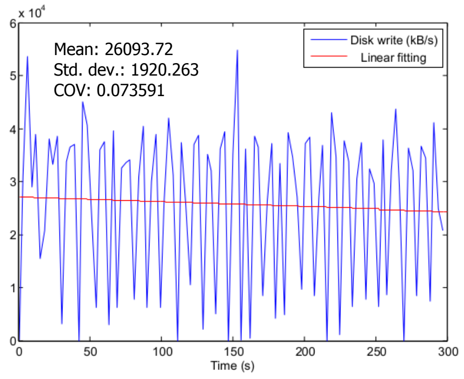


Figure 3. Disk usage of PostMark

Table V
DISK USAGE LEVELS

Level	PostMark
Low	off
High	on

By comparing MySQL's and PostMark's average disk usage, we notice that the former is negligible with respect to the latter, both for low and for high memory usage. Thus, the level of disk usage can be effectively quantified by considering the effect of PostMark alone: the high level of disk usage is implemented by running PostMark while conditioning the environment, whereas the low level is obtained by keeping PostMark turned off. Table V summarizes this implementation.

To implement levels of concurrency we rely on TPC-W. It starts 50 Emulated Browsers which iteratively issue requests to a Web application and stop to "think", the thinking time being a random variable uniformly distributed between 100 and 20,000 ms; the Web application in turn sends requests to the DBMS in order to respond to the browsers. We let the benchmark run for 200 seconds to reach the desired amount of RSS; then, we stop it before submitting the failure-specific workload to implement low level of concurrency; we let it continue while submitting the failure-specific workload to implement high level of concurrency. This is summarized in Table VI.

VI. RESULTS AND ANALYSIS

From the gathered data, we pinpoint three classes of behaviors about failure reproduction: *i)* failures that appear in every execution; *ii)* failures that do not appear in any execution; *iii)* failures that appear in some of the executions.

The observed behavior of the 7 analyzed failures is summarized in Table VII. The 4 failures belonging to the first two groups do not show any measurable dependence of their reproducibility on the factors.

As regards the third group, statistical techniques are employed to expose such a dependence in an objective way. We employ contingency tables to summarize the experimental data and use Fisher's exact test to draw conclusions about the influence of a factor on the occurrence of a failure. Since we neglect the effect of possible interactions between the factors, we build three separate contingency tables per failure, each one being relevant to one of the factors; thereby, when a factor is analyzed, the others' possible effects are attributed to chance. This can be done as the experiment

Table VI
CONCURRENCY LEVELS

Level	TPC-W
Low	stop before submitting failure-specific workload
High	let it run while submitting failure-specific workload

Table VII
OBSERVED CLASSES OF FAILURE REPRODUCTION

Failure	Observed
#32148	never
#38691	sometimes
#38823	always
#42419	sometimes
#44521	never
#46539	sometimes
#55421	always

is perfectly balanced: fixed the level for the factor under consideration, the executions are equally distributed among the combinations of levels for the other factors.

The failures #38691, #42419 and #46539 showed a random behavior. The contingency tables from VIII up to XVI summarize the output data and list the three p-values for the interactions between the failures and each of the factors.

In our context, a p-value is the probability of obtaining the observed or a higher difference in the proportion of failures between the factor's levels, assuming that there is no influence of the factor on the proportion (null hypothesis of Fisher's test). The lower the p-value, the higher our confidence in rejecting the null hypothesis. Two-sided p-value is used to infer the presence of a correlation between a factor's level and the outcome variable; left and right p-values are used to statistically determine if the proportion of failure occurrences is higher respectively with a low or a high level of the factor under consideration.

We notice a strong dependence of the failures #38691 and #42419 on disk usage. As regards the latter, we even see that the failure appears in the totality of the executions with high disk usage: this means that the failure is reproduced deterministically if disk is highly stressed. To confirm this, we ran 5 more executions in each of the treatments with high disk usage and the failure appeared in every iteration. The dependence of these two failures on disk usage can be asserted with a significance less than 0.01, as the p-values are very low. The failure #46539 depends notably on level of concurrency. The dependence can be assumed with a significance of 0.05, as the two-sided p-value is 0.0252. Moreover, we can notice that in the three cases in which a dependence is shown, the right p-values are lower than 0.025. Thus, we can assert that the probability of failure reoccurrence is higher when the respective influential factors are set to a higher level of stress. The findings are summarized in Table XVII.

Table VIII
CONTINGENCY TABLE FOR FAILURE 38691 AND MEMORY OCCUPATION

Memory occupation	Failure occurrences		Total
	NO	YES	
Low	10	30	40
High	5	35	40
Total	15	65	80
Left p-value: 0.9583 Right p-value: 0.1258 Two-sided p-value: 0.2515			

Table IX
CONTINGENCY TABLE FOR FAILURE 38691 AND DISK USAGE

Disk usage	Failure occurrences		Total
	NO	YES	
Low	14	26	40
High	1	39	40
Total	15	65	80
Left p-value: 1.0000 Right p-value: 0.0001* Two-sided p-value: 0.0003*			

Table X
CONTINGENCY TABLE FOR FAILURE 38691 AND LEVEL OF CONCURRENCY

Level of concurrency	Failure occurrences		Total
	NO	YES	
Low	6	34	40
High	9	31	40
Total	15	65	80
Left p-value: 0.2839 Right p-value: 0.8742 Two-sided p-value: 0.5679			

Table XI
CONTINGENCY TABLE FOR FAILURE 42419 AND MEMORY OCCUPATION

Memory occupation	Failure occurrences		Total
	NO	YES	
Low	3	37	40
High	5	35	40
Total	8	72	80
Left p-value: 0.3559 Right p-value: 0.8683 Two-sided p-value: 0.7119			

Table XII
CONTINGENCY TABLE FOR FAILURE 42419 AND DISK USAGE

Disk usage	Failure occurrences		Total
	NO	YES	
Low	8	32	40
High	0	40	40
Total	8	72	80
Left p-value: 1.0000 Right p-value: 0.0027* Two-sided p-value: 0.0053*			

Table XIII
CONTINGENCY TABLE FOR FAILURE 42419 AND LEVEL OF CONCURRENCY

Level of concurrency	Failure occurrences		Total
	NO	YES	
Low	2	38	40
High	6	34	40
Total	8	72	80
Left p-value: 0.1317 Right p-value: 0.9716 Two-sided p-value: 0.2633			

Table XIV
CONTINGENCY TABLE FOR FAILURE 46539 AND MEMORY OCCUPATION

Memory occupation	Failure occurrences		Total
	NO	YES	
Low	6	34	40
High	6	34	40
Total	12	68	80
Left p-value: 0.6223			
Right p-value: 0.6223			
Two-sided p-value: 1.0000			

Table XV
CONTINGENCY TABLE FOR FAILURE 46539 AND DISK USAGE

Disk usage	Failure occurrences		Total
	NO	YES	
Low	7	33	40
High	5	35	40
Total	12	68	80
Left p-value: 0.8259			
Right p-value: 0.3777			
Two-sided p-value: 0.7555			

VII. DISCUSSION

A. Interpretation of results

The performed experiments give an affirmative response to the first research question. A characterization of failures based on the impact of environmental conditions is possible and catches real phenomena regarding their reproducibility. Results show that the reproducibility of some of the failures is actually influenced by the state of the environment. It is possible to isolate environmental components that individually affect the reproducibility. Our proposal for environmental factors reveals to be a good guess, as a strong influence of disk usage and level of concurrency has been highlighted.

As for the second question, we have obtained indications that environmental factors can be conditioned so as to improve detectability of a failure, or even to make its reproduction deterministic. A remarkable case is failure #42419. It has shown up in all executions with high level of disk usage, although randomly observed with low disk usage (see Table XII). This observation supports the claim of the possibility to obtain even a deterministic reproduction by forcing some environmental factors to be in a certain state. It is useful for the purpose outlined in Section I: a debugger can be suggested to systematically set disk usage to a high level, as we did in our experiments.

Table XVI
CONTINGENCY TABLE FOR FAILURE 46539 AND LEVEL OF CONCURRENCY

Level of concurrency	Failure occurrences		Total
	NO	YES	
Low	10	30	40
High	2	38	40
Total	12	68	80
Left p-value: 0.9984			
Right p-value: 0.0126*			
Two-sided p-value: 0.0252*			

Table XVII
FINDINGS SUMMARY

Failure	Most influential factor	Significance
#38691	Disk usage	0.01
#42419	Disk usage	0.01
#46539	Level of concurrency	0.05

Probability of failure reproduction is higher when the level of the relevant influential factor is high, with significance 0.025

Failure #42419 appeared in every execution with high level of disk usage

The overall results suggest that the probability of failure reoccurrence increases if the environmental factors affecting it are set to a high level of stress (see Tables IX and XVI). This observation is reasonable, considering that in a stressed environment rare conditions that can lead to a failure are more likely to occur.

More specifically, let us take the failures #38691 and #46539 as examples. The former is due to a fault in the cleanup operations that are performed on internal data structures when a join operation attempts to request a lock on a table and it is refused. When the disk usage is high, it is more likely that the lock is refused and the data structures corrupted. The failure #46539 is observed if an operation `INSERT IGNORE` is rolled back due to a deadlock or an expired timeout, and the system tries to ignore a non-ignorable error. Deadlocks or timeouts appear more frequently with many requests pending on the DBMS (high level of concurrency).

In Tables X, XI and XIII the opposite trend is observed, that is the proportion of failures appears to be higher for a low level of the relevant factor's stress. However, the difference of proportion is too small to make these observations statistically significant. The recorded p-values are indeed higher than 0.1, so no influence of the relevant factor can be inferred in those three cases.

B. Threats to validity

The validity of a case study has to be analyzed under four aspects: construct validity, internal validity, external validity and reliability [27].

The analysis of construct validity is concerned with determining if the employed techniques actually enable us to observe the property we are intended to investigate. In this context, an issue arises with respect to the implementation of memory occupation. By monitoring it during the experiment, we noticed that in some executions the profiles in Figure 1 were not matched; thereby, an existing influence of this factor may have been concealed.

Furthermore, threats may arise from the use of two distinct virtual machines on the same physical one. On the one hand it mitigates the biasing effect due to having the client script and the system under test on the same physical machine: the memory spaces of the two machines are isolated and the virtual disks are stored in two separate files. On the other hand a source of bias may be the sharing of one physical disk by both machines: to minimize this effect, the scripts on the client machine are devised so that their disk accesses are negligible with respect to the ones of the server machine.

Internal validity is threatened if the outcome variable is affected by factors not taken into consideration. The presence of a class of failures which we were not able to reproduce (see Section VI) is an indication that such factors exist in our experiment.

External validity is concerned with the possibility to generalize our findings. Our sample is too small to draw general conclusions about the relationship between failure reproducibility and environment. Nonetheless, our goal was to prove the possibility to isolate the single environmental factors affecting specific failures.

Reliability is concerned with the extent to which the experiment could be repeated with the same results. As regards this aspect, we notice that the pinpointed relationships are asserted with a very low level of significance; thus, their evidence is strong and there is high likelihood that a researcher repeating the experiment draw the same findings.

VIII. Conclusion and future work

We studied the possibility to characterize software failure reproducibility in terms of the environmental factors by which it is affected. In our experiments with MySQL, two factors, disk usage and level of concurrency, have been discovered to be configurable for an improvement of some failures' reproducibility. Two failures did not appear in any execution of our experiments. This proves the incompleteness of our classification: there may be environmental factors affecting reproducibility that have not been identified. This opens the way to further studies on environmental factors, so as to obtain a complete characterization in this sense.

Our observations are limited to the MySQL case study. Similar studies conducted on different systems may help to draw more general conclusions about the relationship between the failure reproduction process and the environment.

Acknowledgment

This work has been partially supported by the PRIN Project "TENACE" (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

Duke research was supported in part by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) under the JPL subcontract #1440119.

This work has been partially supported by the Spanish National Institute of Information Technologies (INTECO) accordingly to the rule 19 of the digital confidence plan (Digital Agency of Spain) and the Universidad de Leon under the contract X43.

References

[1] J. Gray, "Why Do Computers Stop And What Can Be Done About It?" 1985.

[2] M. Grottke, A. Nikora, and K. Trivedi, "An empirical investigation of fault types in space mission system software," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, 2010, pp. 447–456.

[3] S. Wagner, "Defect classification and defect types revisited," in *Proceedings of the 2008 workshop on Defects in large software systems*. ACM, 2008, pp. 39–40.

[4] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *Software Engineering, IEEE Transactions on*, vol. 18, no. 11, pp. 943–956, 1992.

[5] R. Chillarege and K. A. Bassin, "Software Triggers as a Function of Time - ODC on Field Faults," in *Dependable Computing and Fault-Tolerant Systems*, vol. 10. IEEE Computer Society, 1995.

[6] R. Chillarege and K. Ram Prasad, "Test and development process retrospective-a case study using ODC triggers," in *Dependable Systems and Networks (DSN), 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 669–678.

[7] M. Butcher, H. Munro, and T. Kratschmer, "Improving software testing via ODC: Three case studies," *IBM Systems Journal*, vol. 41, no. 1, pp. 31–44, 2002.

[8] A. Dubey, "Towards adopting ODC in automation application development projects," in *Proceedings of the 5th India Software Engineering Conference*. ACM, 2012, pp. 153–156.

[9] R. B. Grady, "Software failure analysis for high-return process improvement decisions," *Hewlett Packard Journal*, vol. 47, pp. 15–24, 1996.

[10] "IEEE Standard Classification for Software Anomalies," *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, pp. 1–23, Jan 2010.

[11] M. Grottke and K. Trivedi, "Fighting bugs: remove, retry, replicate, and rejuvenate," *Computer*, vol. 40, no. 2, pp. 107–109, Feb 2007.

[12] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. Trivedi, "Fault triggers in open-source software: An experience report," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pp. 178–187.

[13] M. Grottke and K. S. Trivedi, "A classification of software faults," *Journal of Reliability Engineering Association of Japan*, vol. 27, no. 7, pp. 425–438, 2005.

[14] S. Chandra and P. M. Chen, "Whither generic recovery from application faults? a fault study using open-source software," in *Dependable Systems and Networks (DSN), 2000. Proceedings. International Conference on*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 97–106.

[15] I. Lee and R. Iyer, "Software dependability in the Tandem GUARDIAN system," *Software Engineering, IEEE Transactions on*, vol. 21, no. 5, pp. 455–467, 1995.

[16] J. Gray, "A Census of Tandem System Availability Between 1985 and 1990," in *Reliability, IEEE Transactions on*, vol. 39. IEEE, 1990, pp. 409–418.

[17] J. Alonso, M. Grottke, A. Nikora, and K. Trivedi, "An empirical investigation of fault repairs and mitigations in space mission system software," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, June 2013, pp. 1–8.

[18] K. Trivedi, R. Mansharamani, D. S. Kim, M. Grottke, and M. Nambiar, "Recovery from Failures Due to Mandelbugs in IT Systems," in *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*, pp. 224–233.

[19] A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo, "On the Aging Effects Due to Concurrency Bugs: A Case Study on MySQL," in *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*, pp. 211–220.

[20] R. Syed, B. Robinson, and L. Williams, "Does hardware configuration and processor load impact software fault observability?" in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pp. 285–294.

[21] P. Fonseca, C. Li, V. Singhal, and R. Rodrigues, "A study of the internal and external effects of concurrency bugs," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, June 2010, pp. 221–230.

[22] D. C. Montgomery, *Design and analysis of experiments*, 5th ed. Wiley New York, 1997.

[23] A. Brouwer, "The Linux kernel: Memory," [Online] http://www.win.tue.nl/~aeb/linux/lk/lk-9.html, February 2003, Accessed on June 1, 2014.

[24] Oracle Corporation, "The InnoDB Storage Engine," [Online] http://dev.mysql.com/doc/refman/5.1/en/innodb-storage-engine.html, 2014, Accessed on June 1, 2014.

[25] Transaction Processing Council, "TPC-W Standard Specification v. 1.0," [Online] http://www.tpc.org/tpcw/spec/tpcw_v101.pdf, February 2000.

[26] P. Menage, "cgroups documentation," [Online] https://www.kernel.org/doc/Documentation/cgroups/, Accessed on June 1, 2014.

[27] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.