Stress Testing With Influencing Factors to Accelerate Data Race Software Failures

Kun Qiu, Zheng Zheng, *Senior Member, IEEE*, Kishor S. Trivedi, *Life Fellow, IEEE*, and Beibei Yin

***Abstract*—Software failures caused by data race bugs have always been major concerns in parallel and distributed systems, despite significant efforts spent in software testing. Due to their nondeterministic and hard-to-reproduce features, when evaluating systems' operational reliability, a rather long period of experimental execution time is expected to be spent on observing failures caused by data race conditions. To address this problem, in this paper, we make two contributions. First, this paper proposes stress testing with influencing factors, in which the system runs under certain workloads for a long time with controlled stress conditions to accelerate the occurrence of data race failures. Second, it explores and formulates mathematical relationship models between data races' statistical characteristics of time to failure (TTF) or mean TTF (MTTF) and the influencing factors. Such relationship models are used for TTF/MTTF extrapolation under different operational conditions and are essential to reduce systems' reliability evaluation time. The proposed method is empirically evaluated on six applications suffering from failures caused by real-world data race bugs. Through analysis of the experimental results, we obtain several important findings: First, the reduction in the manifestation time to data race failures achieved by controlling the influencing factors is statistically significant. Second, *Power model* is the best-fitting model of the relationship between the MTTF and the influencing factors. Third, *Power Weibull distribution* is the best-fitting probability distribution between the TTF and the influencing factors. Finally, the TTF/MTTF can be accurately estimated with the approach proposed in this paper.**

***Index Terms*—Data race software failure, influencing factors, relationship model, software reliability, software stress testing, time to failure/mean time to failure (TTF/MTTF) estimation.**

NOMENCLATURE

Acronyms

AD	Anderson–Daling test.
ALT/ADT	Accelerated life/degradation testing.
cdf	*s*-cumulative distribution function.
OS	Operating system.
pdf	*s*-probability density function.
ANOVA	Analysis of variance.
CI	*s*-confidence interval.
CL	Concurrency level variable.
DOE	Design of experiments.
GoF	Goodness of fit.
ML	Memory limitation variable.
mRSS	Maximum resident set size.
MTTF	Mean time to failure.
RAM	Random access memory.
TTF	Time to failure.

Notations

α	Level of *s*-significance.
β or σ	The shape parameter.
β_0	The intercept parameter.
β_1	The coefficient parameter.
η	The scale parameter.
λ	The rate parameter.
μ	The location parameter.
$\exp(\cdot)$	The exponential function.
$f(\cdot)$	The *pdf* function.
$\log(\cdot)$	The natural logarithmic function.
S	Explanatory variable.
T	*s*-expectation of the response variable.

Manuscript received February 8, 2018; revised June 19, 2018 and October 14, 2018; accepted January 4, 2019. Date of publication February 14, 2019; date of current version March 2, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61772055 and Grant 61872169, in part by the Equipment Preliminary R&D Project of China under Grant 41402020102, and in part by the Technical Foundation Project of Ministry of Industry and Information Technology of China under Grant JSZL2016601B003. The work of K. S. Trivedi was supported in part by the US NSF under Grant CNS-1523994 and in part by the IBM under a Faculty Grant. Associate Editor: I. Gashi. *(Corresponding author: Zheng Zheng.)*

K. Qiu, Z. Zheng, and B. Yin are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China (e-mail: qiukun@buaa.edu.cn; zhengz@buaa.edu.cn; beibeiyin@buaa.edu.cn).

K. S. Trivedi is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: ktrivedi@duke.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2019.2895052

I. INTRODUCTION

DATA race software failures are increasingly becoming a vital factor in determining the reliability of both small- and large-scale systems. For a mission-critical system, such as JPL/NASA space mission on-board system [1], data race failures can lead to significant threats and consequences to public safety. Software failures are caused by data race bugs introduced during parallel (or distributed) programming practices and can be activated when two or more tasks or threads are concurrently accessing the shared resources while at least one of them is performing writing (or modifying) operations in unexpected orders [2], [3]. Since the execution orders of threads are not certain, a data race failure is usually nondeterministic and hard to reproduce even with the same workloads. To cause a data race failure as soon as possible, data race bug detection and testing techniques have been proposed [2], [4]–[8], aiming to systematically control the scheduler to cover all scheduling

possibilities. However, even with the help of the developed tools, the extensive application of these techniques on large systems is hindered by issues with the time overhead. Therefore, how to handle potential data race failures and improve systems' reliability/availability remains an important research topic.

Fault mitigation techniques, such as the recovery methods [9], are always introduced in real safety-critical systems to make the potential failure less severe and to improve the systems' reliability/availability. Due to the elusive behavior of data race software failures, the practices of fault mitigation and quantitative reliability analysis tend to be proceeded by distinguishing them from deterministic and easy-to-reproduce cases. Grottke *et al.* divided bugs into bohrbugs, which can be certainly reproduced, and mandelbugs, which have nondeterministic manifestation behaviors, and proposed distinguished fault mitigation methods [9]–[12]. As a typical example of mandelbug, data race bugs could be mitigated by the recovery operation schemes, such as reconfiguring the environmental conditions and retrying the fault inputs. Stochastic process models, such as continuous Markov chain (CTMC), semi-Markov process (SMP), Petri Net, etc., are used to assess the quantitative reliability/availability of a system considering potential data race failures and mitigation techniques since these models can easily describe complex behaviors [12], [13].

Solving and optimizing these stochastic process models highly rely on the time to failure (TTF) or mean TTF (MTTF) metric to be known in the first place [12], [14], [15]. TTF (or MTTF) denotes the (average) time, in which an application can normally run for its given workloads. However, the task of estimating the TTF/MTTF is impractical for data race software failures because the failure observation time is too long to collect sufficient samples. Therefore, how to accelerate the data race failure and reduce the TTF/MTTF estimation time is an important topic to be studied.

To the state of the art, the stress testing method, such as accelerating life testing (ALT), has been successfully used for reducing the TTF/MTTF estimation time when a system suffering from aging-related bugs (ARBs) [16]–[18]. The stress testing method uses the TTF/MTTF data collected under stressed conditions to predict the one under target conditions according to certain models. Facing the same problem, the feasibility of using the stress testing method to deal with failures caused by data race bugs has not been systematically studied. Two critical issues need to be explored to facilitate the stress testing method as follows.

1) How to accelerate the process of detecting data race software failures? We need to determine the influencing factors as the stress condition to reduce data race failures' manifestation time and reduce the essential time.
2) How to effectively and efficiently estimate the TTF or MTTF caused by data race? We need to determine a model that can relate the TTF/MTTF under different stress conditions.

In this paper, we present an empirical study of using stress testing with influencing factors to reduce the TTF/MTTF estimation time. Stress testing is commonly conducted to flush out data race conditions by ceaselessly performing certain workloads for a long period of time [5]–[7]. However, according to existing research [5] and our testing experiences, we find that the purely stress testing practices are neither efficient nor effective because most of schedule possibilities are still not covered after iterating workloads for many loops. Aiming to overcome deficiencies of stress testing, we additionally control programs' external executing environments, known as influencing factors, to increase the possibilities of encountering data race preconditions. Based on the software failure mechanism of data races, we select three influencing factors that could influence programs' internal execution processes and then increase the risk of triggering the data race bugs.

Furthermore, we empirically study the relationship models between TTF/MTTF and the influencing factors. The relationship model is a quantified strength function that relates the response variable, i.e., TTF/MTTF, with the explanatory variable, i.e., the influencing factor [19], [20]. With this relationship model, the stressed conditions' TTF/MTTF can be extrapolated back to predict other operational (or nonstressed) condition values. Considering that the relationship model plays a pivotal role in solving the proposed issues, we employ a regression analysis method to delve into its formula in this paper.

A methodology including experiments and regression analysis is presented to study the influences and the relationship model for the proposed influencing factors. This methodology combines the design of experiments (DOE) [21] and the stress testing analysis [19], [20]. We conduct the methodology on six different desktop/server applications, which suffer from real data race failures.

This paper reveals seven interesting findings that provide useful guidelines for software data race testing, software reliability modeling, and evaluating, as shown in the following.

1) *Finding #1:* All three proposed influencing factors can be used in the stress testing to reduce the expected time to data race failure for the experimental applications. For the applications we tested, we could save the reproducing time for at least 15.9 times. For example, we did not manifest a failure hidden in an application known as *Pbzip* when retrying the failure workload 50 000 times but did observe the failure with an average of 628.2 reattempts.
2) *Finding #2:* The stress testing condition, ⟨*High* memory limitation, *High* concurrency level, *Multiple* parallel level⟩, is the optimized condition to minimize the time to data race failure.[1] *Findings #1 and #2* indicate a lightweight probabilistic stress testing method to detect and reproduce race conditions during development phrase.
3) *Finding #3:* The MTTF varies with different influencing factor settings. In addition, the variations caused by different factors for the same data race suffering application are different. This finding gives the formal statistical evidence that the *environmental diversity* mitigation method proposed in [11] and [12] can be applied for data race failures. *Environmental diversity* approach expects to im-

[1]The terms in this finding are defined in Section II.

prove systems' availability by reconfiguring applications' environmental conditions and retrying the failure inputs.

4) *Finding #4: Power model* best fits the relationship between MTTF and the influencing factor explanatory variable.[2]
5) *Finding #5: Weibull distribution model* best fits the TTF distribution model of data race failures.[3]
6) *Finding #6: Power Weibull distribution model* best fits the relationship model between TTF and the influencing factor explanatory variable.
7) *Finding #7:* The MTTF and TTF can be accurately estimated by using the proposed influencing factors as explanatory variables. *Findings #4–#7* provide the TTF/MTTF distribution and estimation information, which are necessary for performing quantitative reliability/availability analysis and designing fault mitigation techniques.

This paper is organized as follows. Section II presents the influencing factors for data race software failures. Section III presents the experimental methodology employed in this paper. Section IV organizes the experimental data analysis results and seven findings into the answers to three research questions. Sections V lists potential threats to the validity of this paper. Section VI shows the related work. Finally, conclusions and future work are given in Section VII.

II. Influencing Factors

In this section, we describe the influencing factors and discuss how the influencing factors affect the occurrence of data race failures. The factors are expected to accelerate data race failures by impacting the failure manifestation process. Recall that a data race occurs when there are memory accesses in a program that satisfy all the following conditions:

1) operating on the same memory location;
2) performing concurrently or in parallel;
3) writing (or modifying) operations;
4) not protected by synchronization mechanisms.

Therefore, the main idea of influencing a program's internal execution and accelerating the occurrence of data race is to increase the risk of breaking up unprotected synchronization of writing (or modifying) operations on the shared locations. Based on this motivation, we select three influencing factors as follows.

1) Memory limitation, which is the maximum resident set size (*mRSS*) of physical memory available for the running program.
2) Concurrency level, which is the number of concurrent visiting clients/users of the program.
3) Parallel level, which is the number of processors available for the program.

In the following, we will explain how these three influencing factors affect the internal execution behavior of a program and increase the probability of triggering data race conditions.

[2]The mathematical format of *Power model* is defined in Section III-B3.

[3]The *pdf* of *Weibull distribution* and *Power Weibull distribution* are given in Section IV-C.

A. Memory Limitation

The memory limitation controls the maximum physical memory set size a to-be-tested application can access. The physical memory limitation can increase the frequency of thread context switching [22] and, then, indirectly increase the threads' risk of being interrupted from the synchronization parts. In a thread context switch, the accessing of a processor is switched from one thread to another. According to the principle of an operating system (OS) [22], [23], if a processor tries to read from a virtual memory address that is not currently mapped to a RAM address, a page fault occurs; then the running thread is pre-empted and suspended to the waiting state, during which its required data is mapped into the RAM from the disk by the OS. Simultaneously, another thread in the ready state starts to run. Thus, if the available physical memory is limited, the frequency of page fault is increased, thereby increasing the interaction probability among concurrent threads, which increases the probability of encountering unprotected synchronization operations. Therefore, we select memory limitation as the first influencing factor.

Memory limitations are an indirect and lightweight method to control the frequency of context switches. The alternative method to this aim is inserting instruments in threads/processes and scheduling them deliberately. Such intrusive approach could slow down the system and faces space explosion problems, hence is hard to be employed for large scaled applications. However, too harsh memory limitation stress could lead to the thrashing problem [24], in which pages kept on swapping in and out of RAM alternatively. Thrashing could lead to longer workload execution time and fail to reduce the testing time. Therefore, we must reserve a certain margin so that thrashing has no significant impact on the experimental time.

B. Concurrency Level

The concurrency level refers to the number of different kernel threads or processes to be executed out-of-order or in partial order based on which user-level tasks are mapped. According to our prior experimental experience and literature studies [25], [26], data race software failures occur more frequently with higher concurrent workloads. Higher concurrency level workloads increase the complexity and intensity of context switching for racing the shared resources. Therefore, they could increase the possibility of hitting certain thread orders that can break the synchronization and thus trigger a data race failure. Due to the aforementioned reasons, the concurrency level is selected as the second influencing factor in this paper.

C. Parallel Level

The parallel level controls the number of processors that can be accessed by the testing application. A data race occurs when two or more threads are executing modifications on a shared memory location without synchronization. For a single-processor-equipped system, the threads are scheduled to alternatively access the processor, and the shared memory location is accessed by only one threat at a time. While for a multiple-processor case, the spawned threads can run on

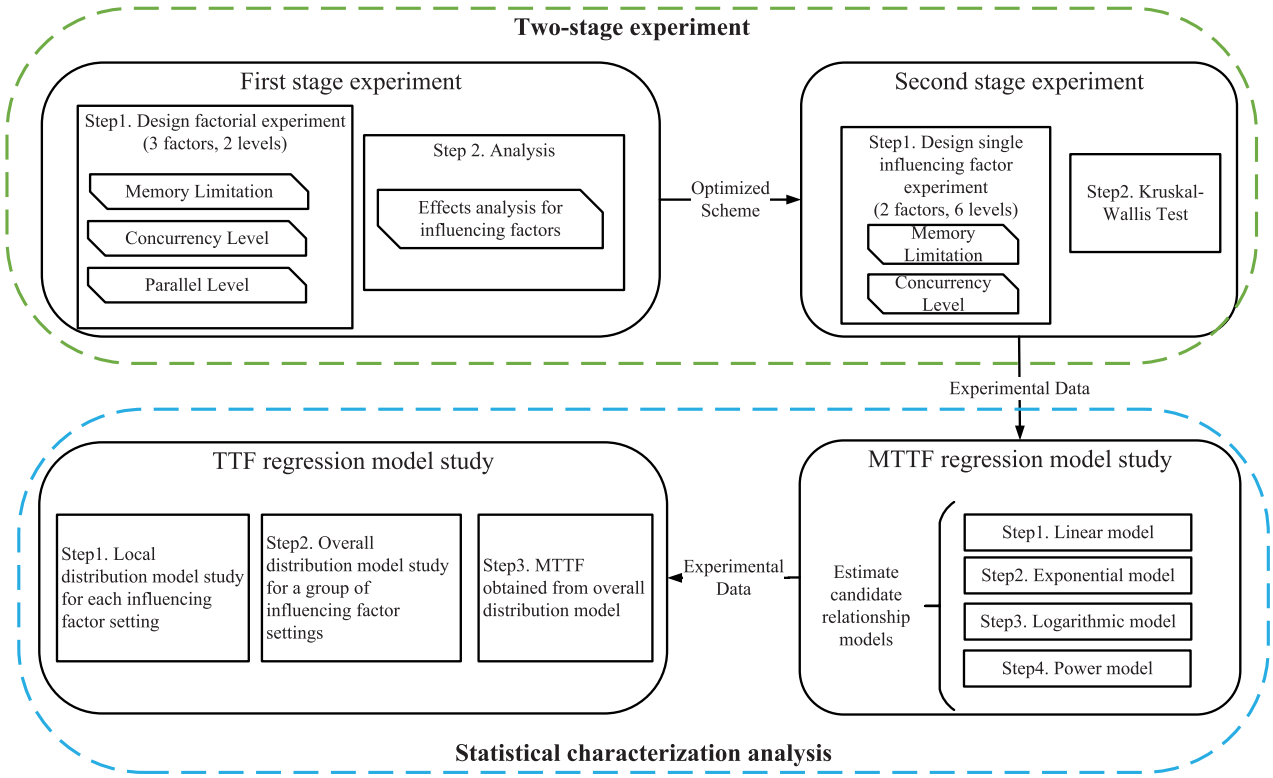

Fig. 1. Overview of the experiment and analysis process.

multiple processors simultaneously, and the probability of two or more threads modifying the shared memory location at the same time is greatly increased. Compared with the multiple-processor case, it is easier for a single-processor system to maintain the synchronization of modification operations because only one thread can access the location in each epoch. By contrast, if multiple processors are equipped, the risk of hitting certain orders among the threads is increased due to the possibility of more threads attempting to modify the shared location. Considering this characteristic, we select parallel level as the third influencing factor.

III. Experimental Study

In this section, a stress testing experimental methodology is presented to explore the impacts of the influencing factors proposed for data race software failures and their mathematical relationship models. In addition, we adopt the methodology for six applications that suffer from real data race software failures.

A. Research Questions

The aim of the experiments is to answer the following research questions on influencing factors.

- *RQ1:* Whether the influencing factors have impacts on the MTTF?
- *RQ2:* What is the relationship model between the MTTF and the influencing factors?
- *RQ3:* What is the relationship model between the TTF and the influencing factors?

We propose *RQ1* to empirically verify the effectiveness of the proposed influencing factors on accelerating the occurrence data race failures. *RQ2* and *RQ3* are proposed to figure out the models for the purpose of making predictions efficiently.

B. Overview of the Experimental Design and Analysis Process

Fig. 1 shows the architecture of our experimental design and analysis process. The process includes two main phases, the experiment phase and the statistical characterization analysis phase. The experiment phase includes two stages. The purpose of the first stage is to determine which influencing factor(s) or their interactions can be adopted to reduce the testing time. The purpose of the second stage is to collect sufficient samples to study the mathematical relationship models between TTF/MTTF and the influencing factors. When the first stage is accomplished, the optimized conditions of the influencing factors, which are expected to minimize the testing time, are obtained to reduce the time costs of the second stage of the experiment. Then, after the data are collected in the second stage, regression analysis is applied to explore the relationship between TTF (or MTTF) and the influencing factors. In the following, we explain the details of the experimental process and the data analysis by four parts.

1) First Stage of the Experiment: We adopt the full factorial DOE [21] to learn the *s*-effects of three influencing factors, namely memory limitation, concurrency level, and parallel level, and their interactions. The objective of a full factorial experiment is to create a set of test scenarios (namely, a test plan) that can separate out the impact of specific variables from that of a group

TABLE I
EXPERIMENTAL PLAN FOR THE FIRST STAGE

Index	Memory	Concurrency	Parallel	# of Replicates
EXP1	Low	Low	Single	5
EXP2	Low	Low	Multiple	5
EXP3	Low	High	Single	5
EXP4	Low	High	Multiple	5
EXP5	High	Low	Single	5
EXP6	High	Low	Multiple	5
EXP7	High	High	Single	5
EXP8	High	High	Multiple	5

of variables. In the factorial experiments, we consider two levels for each factor.

Table I shows the plan for the first stage of the experiment. The second to fourth columns denote the stress levels of the influencing factors. The last column represents the number of replicates of each stress condition. Since there are two levels for each factor, a total of 2^3 (8) combinations are required to be tested. We use *High* and *Low* to denote the high stressed level and low stressed level for memory limitation and concurrency level, and use *Multiple* and *Single* to denote multiple level and single level for a parallel level factor. Taking into account the statistical error, in this stage, the experiment under each stress condition is repeated five times, which is calculated according to the algorithm in [21], for a total of 8×5 (40) replicates for a given application.

Based on the data obtained in the first stage of the factorial experiment, the *s*-effects [21] of each influencing factor and their interactions are analyzed to determine which influencing factors have significant impacts on accelerating the occurrence of data race failures. We conclude that an influencing factor has significant *s*-effects if its absolute value is greater than the $\alpha = 0.05$ significant value obtained by Lenth's method [27]. The optimized conditions are determined by comparing the means of the experimental results under the eight stressed conditions. The analysis results are shown in Section IV-A.

2) Second Stage of the Experiment: We adopt the single-factor DOE [21] to learn the relationship model between TTF/MTTF and one of the two influencing factors, i.e., memory limitation and concurrency level. Through the relationship model, we can obtain the rate of change of the TTF/MTTF with respect to influencing factor settings and then can predict the TTF/MTTF in other target settings. Because the applications we focused on are not high-performance-computing (HPC) applications, which perform complex algorithms on huge data sets and usually need dozens of processors in different clusters [28]; thereby, most of the applications are running on a single computer machine with a discrete, limited processors, in contrast to other factors, which are continuous and have wide ranges of values. It is infeasible to employ the number of processors as an explanatory variable for TTF/MTTF. Therefore, the relationship model for the parallel level factor is not explored in this paper.

The stress loading strategy, including the number of stress levels, sample sizes, and settings for each level, is in the next step. In this paper, for each factor, we design 6 stress levels and perform 15 replicates for each level, for a total of $2 \times 6 \times 15$ (180) replicates for each application. Because the settings for each level are dependent on the applications, we explain the values of the settings in Section III-D along with the experimental applications.

For the data obtained in the second stage of the experiment, we first use the nonparametric ANOVA, in terms of the Kruskal–Wallis test [29], to determine whether the distribution of TTF changes with the settings of the memory limitation or concurrency level. We conclude that the distribution changes significantly if the *p-value* of the Kruskal–Wallis test is less than $\alpha = 0.05$. Moreover, the specific variation function is explored in the regression analysis phase.

3) MTTF Regression Model Study: We apply least-squares regression method to analyze the relationship model between the MTTF variable and the influencing factor explanatory variables. Four types of candidate relationship models, namely *Linear model*, *Exponential model*, *Logarithmic model*, and *Power model,* [19], [20], [30], [31], are tested to determine the best-fitting model by comparing their goodness of fit (GoF) statistics. According to the central limited theorem [32], the mean of a large number of independent and identically distributed random variables tends to follow a *Normal* distribution. Therefore, in the regression analysis, we assume that the MTTF follows a *Normal* distribution. The analysis details will be described in Section IV-B.

The mathematical forms of the four candidate relationship models are given as follows:

Linear model: $T = \beta_0 + \beta_1 S$
Exponential model: $T = \beta_0 e^{\beta_1 S}$
Logarithmic model: $T = \beta_0 + \beta_1 \log(S)$
Power model: $T = \beta_0 S^{\beta_1}$

where T denotes the *s*-expectation of the response variable, i.e., the MTTF variable, S denotes the explanatory variable, which is the memory limitation variable or concurrency level variable in this paper, β_0 denotes the intercept parameter, β_1 denotes the coefficient parameter, and $\log(\cdot)$ is the natural logarithm function.

4) TTF Regression Model Study: In this paper, we use the *s*-maximum-likelihood estimation to analyze the relationship between TTF and the influencing factor variables. According to the *s*-maximum-likelihood estimation procedure [19], [20], we first study the local distribution of TTF under a certain influencing factor setting and then study the overall distribution of TTF under a whole group of influencing factor settings. Finally, we calculate MTTF from probability density function (*pdf*) of TTF. The details of the three analysis steps are described in Section IV-C.

C. Experimental Subjects

In this section, we will briefly introduce the software applications used in our experiments.

1) Airline and Account: They are two applications and are widely used as benchmarks for evaluating the validity of data race bug detection techniques [33], [34]. *Airline* provides the service of selling tickets for an airline cooperation. The data race

TABLE II
EXPERIMENTAL APPLICATIONS AND EXPERIMENTAL SETTINGS FOR THE FIRST STAGE

App.	Description	Workload format	Influencing Factor	Low L.	High L.	*mRSS*	Max rep. time
Airline	*Airline* tickets purchase software.	(# of clients,# sold tickets)	Memory Limitation	1G	10,000K	16,584K	10,000
			Concurrency Level	10	60		
Account	Bank accounts management software.	(# of business account, # of personal account)	Memory Limitation	1G	10,000K	15,548K	10,000
			Concurrency Level	2	20		
MySQL	Database management software.	Client with ALTER operation, Client with FLUSH operation	Memory Limitation	1G	20M	35M	10^6
			Concurrency Level	2	30		
Mozilla	*Mozilla*'s JavaScript engine.	Client with garbage collector, Client with context operation.	Memory Limitation	1G	1,500K	4,260K	60,000
			Concurrency Level	10	60		
Pbzip	Parallel version of a file compressor application.	Concurrently compressing a random file with 200K bytes.	Memory Limitation	1G	10,000K	37,360K	10,000
			Concurrency Level	2	2,000		
Memcached	Distributed memory caching sysmtem.	Concurrent clients with increasing certain cached data operations.	Memory Limitation	1G	1,000K	3,498K	10,000
			Concurrency Level	5	50		

bug in *Airline* is caused by an unprotected shared Boolean variable, known as *StopSales*, which indicates whether the tickets are sold out. Similar to *Airline*, *Account* contains an unprotected field in a class, known as *PersonalAccount*, which represents a personal account in a bank. We use the *try-exception* trick to capture the exception caused by the data race bug, which ends the application. To eliminate the possibility of misjudging, we assert the applications' exit number to make sure the failure is caused by data race.

2) MySQL: It is a popular open-source database management software system. We select a workload that can trigger a data race bug indexed by #38691 in the *MySQL* bug report website [35]. The bug is an incomplete implementation of managing locks for multiple tables in the function named *mysql_multi_update_prepare()*. The failure is determined by its ERROR numbers logged in log files, which is 2013.

3) Mozilla: It is downloaded from the test benchmark of *Radbench* [36], and is a part of *Mozilla* SpiderMonkey (the Firefox JavaScript engine). The data race bug can be triggered when the garbage collector releases the resources used by other threads, which cause a null pointer reference crash. As a result, we assert the data race failure by monitoring a segmentation fault [22], whose exit (or error) code is 139.

4) Pbzip: It is the parallel version of *bzip2*, which is a file compressor, and has been studied in the literature [7]. The version number of *Pbzip* tested in this paper is 2-0.9.4. During the execution of the application, concurrent consumer threads are spawned to compress a target file. However, the main thread may release shared resources while its consumer threads are not completed, which cause a segmentation fault crash. Therefore, this failure can be confirmed by the application's exit code, 139.

5) Memcached: It is an open-source, distributed memory object caching system that is intended to increase the speed of dynamic web applications by alleviating database loads. The version we studied is indexed by 1.4.4 [7], [36]. Failure occurs when two or more clients are concurrently modifying a cached item, and one of them may release the item when other threads are using it. For this application, we first start the *Memcached* server and then spawn concurrent clients to increase modification of the cached items. The failure can be asserted by comparing the computed values with the expected values.

D. Experimental Setup

Our experiments are conducted on a virtual machine with 1 GB RAM, the Ubuntu 14.04 OS, gcc 4.8.4, and JDK 1.8.0_91. The hardware system includes an Intel core i7-3770S 3.1 GHz four CPUs, 8 GB RAM. Since many applications and samples are tested, the virtual machine can effectively and efficiently replicate the same testing operating environments for different applications.

According to the experimental procedure shown in Fig. 1, we set up the two stages of the experiments for six applications. Table II summarizes the experimental applications and the experimental settings for the first stage. The second column presents a brief description of the applications' functionalities and features. The third column shows the workload formats required to reproduce the data race failures. The fourth column presents the influencing factors to be utilized. The fifth and sixth columns present the settings of the *High* level and *Low* level. The seventh column shows the *mRSS* [23] obtained for the loaded workloads. *mRSS* indicates that the maximum physical memory is used by performing the given workload. We treat the *mRSS* as the boundary between the *High* stressed and *Low* stressed memory limitation levels. *High* stressed values are expected to be less than the *mRSS* to constrain memory resources and to trigger more page faults. By contrast, *Low* stressed values should be larger than *mRSS* to cause little or no unnecessary page faults. The last column shows the number of maximum workload repetition times.

Table III summarizes the experimental settings for the second stage. The second and third columns show the six stress levels' values, which are determined by experimental analysis of the first stage as described in Section IV-A. The last column shows

TABLE III
EXPERIMENTAL SETTINGS FOR THE SECOND STAGE

App.	Memory limitation	Concurrency level	# of Replicates
Airline	1,000K, 2,000K, 4,000K, 6,000K, 8,000K,10,000K.	2, 5, 10, 15, 20, 25.	15
Account	1,000K, 2,000K, 6,000K, 8,000K,10,000K, 12,000K.	2, 10, 15, 20, 25, 30.	15
MySQL	5M, 10M, 15M, 20M, 25M, 30M.	2, 10, 20, 30, 40, 50.	15
Pbzip	30M, 40M, 50M, 60M, 70M, 80M.	1,000, 1,500, 2,000, 2,500, 3,000, 3,500.	15
Mozilla	1,500K, 1,600K, 1,700K, 1,800K, 1,900K, 2,000K.	10, 20, 30, 40, 50, 60.	15
Memcached	500K, 1,000K, 1,500K, 2,000K, 2,500K, 3,000K.	5, 10, 20, 30, 40, 50.	15

the number of replicates for each stress level. In the following, we further introduce the setup of our experiments in terms of five components.

1) Workload: In a stress testing replicate, we iteratively load a fixed workload to a target application. This strategy has been widely adopted in the testing of data race failures [5], [37]. The workload utilized is provided by the bug reporters or benchmark creators and may cause an observable and automatically detectable failure.

2) Response Variable (or TTF): Instead of using the clock time directly, we use the number of workload repetitions before the manifestation of a data race failure as the response variable to measure the TTF for an application, which can greatly reduce the cost of recording and is widely used in research [16], [17], and [18]. Note that the repetition number and the clock time can reciprocally transform into each other given the average workload performance time.

The observed data for the responsible variable data can be divided into two types. The first type is known as *complete data* [19], which are generated when a test ends upon observing a data race software failure. The second type is called *censored data* [19], which are collected when a test ends without observing a failure before the maximum repetitions are reached.

3) Memory Limitation Settings: It is implemented by limiting the gross physical memory size that a process can access to a predefined value by a tool *cgroup* [38]. To avoid the harsh memory thrashing problem, the memory limitation value should set to be larger than 30% of *mRSS*, which can be obtained by the Linux tool called */usr/bin/time*. As shown in Table II, in the first stage, the *High* stressed level is configured to a value between 30% and 60% of an application's *mRSS*, and the *Low* stressed level is then configured to a value greater than *mRSS* to minimize the effects caused by strained RAM resources. In the second stage, we have a uniform selection of six values around the value representing *High*, as shown in Table III.

4) Concurrency Level Settings: For *Airline*, *Account*, and *Pbzip*, the concurrency level settings can be implemented by setting their input parameters. For *MySQL*, *Mozilla*, and *Memcached*, their concurrency levels are controlled by changing the number of connecting clients. As shown in Table II, in the first stage, the values of the *Low* and *High* levels are set according to the applications' specifications. Since the data race bugs we tested are very hard to reproduce with the concurrency level reported, we then use the number in provided bug reproduction script as the *Low* level, such as 2 or 10. Since too large concurrency could lead to out-of-memory errors and block the OS for a long time, for *High* level, we first select a large number and gradually decrease that number until no out-of-memory errors or blocking problems. At the second stage, we have a uniform selection of six values between the values representing the *Low* level and the *High* level. For example, for *Memcached*, we set the *High* level to 50 and the *Low* level to 5 for the first stage of experiment. Moreover, the settings for its second stage are set to 5, 10, 20, 30, 40, and 50, as shown in Table III.

5) Parallel Level Settings: The factor setting is implemented by configuring the number of processors for a virtual machine. In the first stage, the number of processors is set to 2 for *Multiple* level and 1 for *Single* level.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we use statistical methods to analyze the collected experimental data and attempt to study the following three research questions.

RQ1: Whether the influencing factors have impacts on the MTTF?

RQ2: What is the relationship model between the MTTF and the influencing factors?

RQ3: What is the relationship model between the TTF and the influencing factors?

According to the research questions studied, the analyses are organized into three parts, and seven findings are presented in the process of discussion.

A. Answer for RQ1

For a better explanation, we divide the impact analysis and findings into two parts according to their stages.

1) Analysis and Findings in the First Stage: The s-effects analysis method in the literature [21] was employed to analyze data collected in the first stage (see Table I) for each application. For all to-be-examined factors, we first assume that there is a linear regression model between them and the failure time data. Then, the *s*-effects or coefficients for each factor and their interaction can be calculated by solving a set of linear equations with respect to the eight treatments, whose algorithms can be found in [21]. Second, we need to perform an ANOVA F test to test the hypothesis of whether the *s*-effects are statistically significant for the regression model, and we adopt $\alpha = 0.05$ as the threshold. Note that the linear regression model assumption is just a coarse assumption applied to filter significant influencing factors, and a more precise model will be obtained through the second stage analysis. Fig. 2 illustrates the absolute values of the *s*-effects of both influencing factors and their interactions for six experimental applications. The red dashed reference line

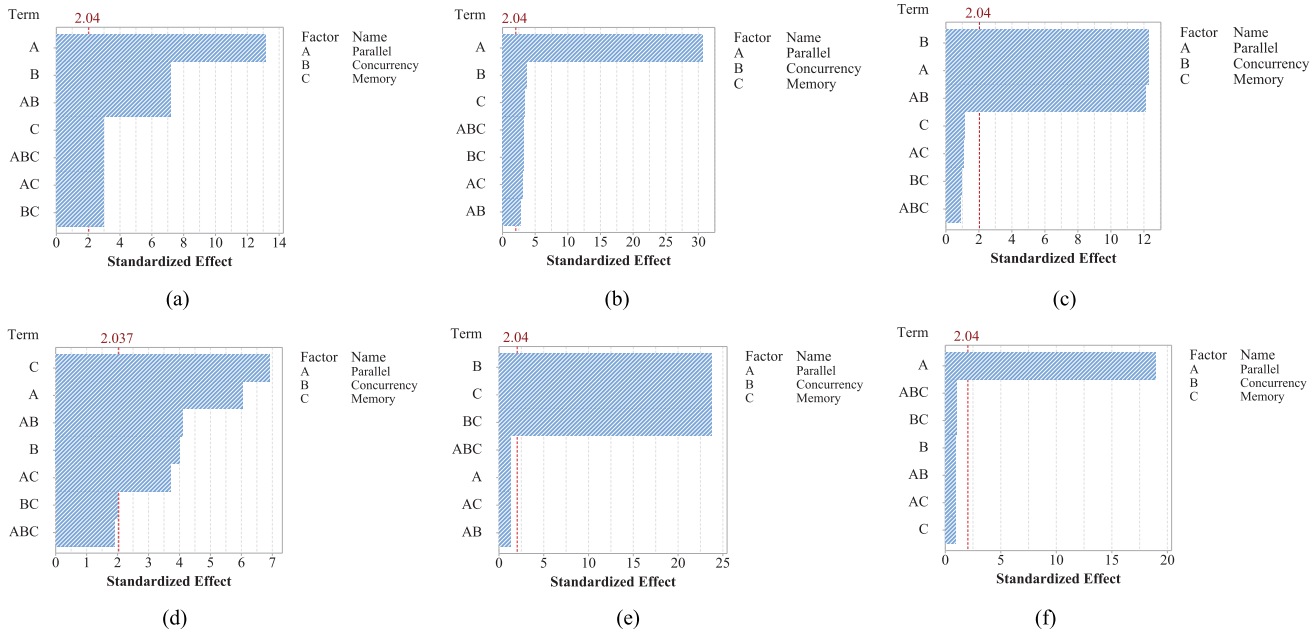

Fig. 2. Absolute values of *s*-effects for both influencing factors and their interactions of six experimental applications. (a) *Airline.* (b) *Account.* (c) *MySQL.* (d) *Mozilla.* (e) *Pbzip.* (f) *Memcached.*

denotes *s*-significant value of $\alpha = 0.05$, indicating whether the factor has a significant impact.

Finding #1: All three proposed influencing factors can be used in stress testing to reduce the expected time to data race failure for the experimental applications.

Fig. 2 shows that at least one of the three influencing factors has a greater absolute *s*-effect values than the significant value for each application. Specifically, parallel level has significant impacts on five of six applications and has the largest magnitude of *s*-effects for *Airline*, *Account*, *MySQL*, and *Memcached* applications, indicating the large effectiveness of the failure time reduction for their data races. Concurrency level has significant impacts on five of six applications and is the factor with the largest *s*-effects on *Account*, *MySQL*, and *Pbzip* applications. Memory limitation affects four of six applications and has the largest *s*-effects on *Mozilla* and *Pbzip* applications.

Additionally, we observe that an influencing factor may not affect a certain application. For example, for *Pbzip* application, the value of *s*-effects of parallel level is around 1.50, which is less than its *s*-significant value, 2.04, indicating that the multiple processors setting does not have a statistically significant impact on the probability of triggering its data race bug. By analyzing the codes or configurations of some applications, we find that the settings of the applications may shield the impact of the influencing factors. For example, the configuration of some applications can implicitly constrain the number of concurrent users allowed, which limits the influence of the concurrency levels. According to the documentation of *Memcached* [39], although its default maximum number of threads can be configured, unexpectedly, it adopts a different user connection service mechanism. One thread can simultaneously serve several connections because of their low memory costs; however, the expectations are that one user connection is handled by one thread/process, such as in the *Apache* server software. The special *Memcached* service pattern can shield the concurrency level factor.

Optimized condition of stress testing was determined by analyzing the mean value of the response variable of the first stage of experiments. Under the optimized condition of stress testing, the time to data race failure is expected to be reduced the most. We tested eight different conditions, known as *EXP1* to *EXP8* for the first stage. Table IV presents the mean values of the response variable under the eight experimental conditions for six applications. The float number in each cell represents the value calculated by (1), and the symbol *(censored)* denotes that no failure is observed under the corresponding stress settings before the maximum repetitions are reached, and the bold font is used to emphasize the smallest value

$$\bar{t}_i = \frac{\sum_j^{n_i} t_{ij}}{n_i} \tag{1}$$

where $\bar{t}_i$ denotes the mean value of the response variable (or MTTF) under the i_{th} stress conditions, n_i denotes the number of observed samples under the i_{th} stress condition with casting off the censored ones, and t_{ij} denotes the observed response variable value for the i_{th} stress condition and the j_{th} replicate.

Finding #2: The stress testing condition, ⟨*High* memory limitation, *High* concurrency level, *Multiple* parallel level⟩, is the optimized condition that minimizes the time to data race failure.

According to Table IV, the mean values of the response variable for *EXP8* are much smaller than those of the other stress conditions. For *Account*, *MySQL*, and *Memcached*, their mean values, which are 144.4, 5.0, and 39.0, respectively, are the smallest among the eight tested conditions. For the other three applications, i.e., *Airline*, *Mozilla*, and *Pbzip*, their corresponding mean values of the response variable for *EXP8*, which are 9.8, 546.8, and 1608.0,

TABLE IV
MEAN VALUES OF RESPONSE VARIABLE IN THE FIRST STAGE OF EXPERIMENTS

App.	*EXP1*	*EXP2*	*EXP3*	*EXP4*	*EXP5*	*EXP6*	*EXP7*	*EXP8*
Airline	*(censored)*	13.8	4838.0	**7.8**	*(censored)*	9.8	38.4	9.8
Account	*(censored)*	505.2	*(censored)*	181.6	*(censored)*	398.6	5291.0	**144.4**
MySQL	*(censored)*	2870.2	4295.8	46.0	*(censored)*	1414.4	55.2	**5.0**
Mozilla	*(censored)*	4129.4	*(censored)*	**104.2**	3266.2	3199.4	3391.4	546.8
Pbzip	*(censored)*	*(censored)*	*(censored)*	*(censored)*	*(censored)*	*(censored)*	**628.8**	1608.0
Memcacehd	*(censored)*	173.0	*(censored)*	51.4	*(censored)*	45.2	*(censored)*	**39.0**

respectively, are the second largest. For experimental condition *EXP1*, all cells are filled with *(censored)* symbols, indicating that no failures are detected for all the tested applications. Because the stress testing condition *EXP1* is ⟨*Low* memory limitation, *Low* concurrency level, *Single* parallel level⟩, which is expected to be the nonstressed working environment, we can conclude that data race failures are difficult to manifest under normal operational conditions. For the other six experimental conditions, i.e., *EXP2–EXP7*, the expected data race failures can be observed before the maximum number of repetitions, indicating that using only one of the three influencing factors for the stress testing can accelerate the occurrence of the data race failures of the applications.

2) Analysis and Findings in the Second Stage: We adopted the nonparametric ANOVA, known as the Kruskal–Wallis test [29], to analyze the data collected from the second stage. We tested two hypotheses for two groups (six levels for each) of data, i.e., memory limitation group and concurrency level group, for each application. If the Kruskal–Wallis test's *p-value* is less than 0.05, we reject the null hypothesis. The two hypotheses are as follows.

1) *Hypothesis 1:* The TTF of each subject is not affected by the different levels of memory limitations.
2) *Hypothesis 2:* The TTF does not change with the variation in concurrency levels.

Table V summarizes the Kruskal–Wallis test analysis results for the two hypotheses. The third column, *N*, denotes the number of replicates observed with casting off the *censored data*. The fourth and fifth columns, *H* and DF, denote the Kruskal–Wallis statistics and the *s*-degree of freedom, respectively. The last column shows the Kruskal–Wallis test's *p-values*.

Finding #3: The MTTF varies with different influencing factor settings. In addition, the variations caused by different factors for the same data race suffering application are different.

According to Table V, it is clear that all the applications reject at least one hypothesis, indicating that their MTTFs are sensitive to the influencing factor settings, i.e., memory limitation and concurrency level. First, for the memory limitation variation study, *Airline*, *Account*, and *Pbzip* applications' *p-values*, which are 0.018, 0.001, and 0.001, respectively, are less than 0.05, indicating that the MTTFs for these three applications vary with the setup of the physical memory set sizes [23]. Second, for the concurrency level variation study, *Account*, *MySQL*, *Mozilla*, *Pbzip*, and *Memcached* applications' *p-values*, which are 0.001, 0.001, 0.001, 0.001, and 0.002, respectively, are smaller than

TABLE V
KRUSKAL–WALLIS TEST ANALYSIS RESULTS

App.	K. S. Test	N	H	DF	*p-value*
Airline	Hypothesis 1	90	13.67	5	0.018
	Hypothesis 2	90	8.3	5	0.140
Account	Hypothesis 1	90	44.94	5	< 0.001
	Hypothesis 2	90	21.54	5	0.001
MySQL	Hypothesis 1	90	4.29	5	0.508
	Hypothesis 2	90	68.54	5	< 0.001
Mozilla	Hypothesis 1	78	3.27	5	0.658
	Hypothesis 2	82	66.13	5	< 0.001
Pbzip	Hypothesis 1	86	27.02	5	< 0.001
	Hypothesis 2	90	57.69	5	< 0.001
Memcached	Hypothesis 1	90	1.97	5	0.853
	Hypothesis 2	90	18.76	5	0.002

0.05, indicating that the MTTFs for these three applications are varying with the concurrent clients/users number setup. Four groups have *p-values* greater than 0.05, which indicates the small variance among different influencing factor settings. This could be caused by some special mechanisms inside the application, which makes the application insensitive to the stress factor. For example, *Memcached* serves multiple clients in one thread. Therefore, having a better understanding of program's internal architecture can help in determining the accelerating significance of an influencing factor.

Fig. 3 shows the boxplot of TTF (i.e., response variable) by the setup of corresponding influencing settings. A total of eight sensitive groups have *p-values* less than 0.05, and there are four nonsensitive groups. We can observe that in memory limitation subfigures, i.e., Fig. 3(a), (c), and (i), there are obvious upward trends in the TTF with respect to the increasing of memory limitation settings. By contrast, for subfigures illustrating the impacts of concurrency level, i.e., Fig. 3(d), (f), (h), (j), and (l), there are recognizable downward trends with increasing concurrency level. Moreover, Fig. 3(b), (e), (g), and (k) shows flatter variations with different influencing settings, and we treat them as nonsensitive groups. Based on these subfigures, we obtain an overall, but rough, understanding of the relationship between response variable and influencing factors. In the following two sections, we present the analysis procedures and findings for

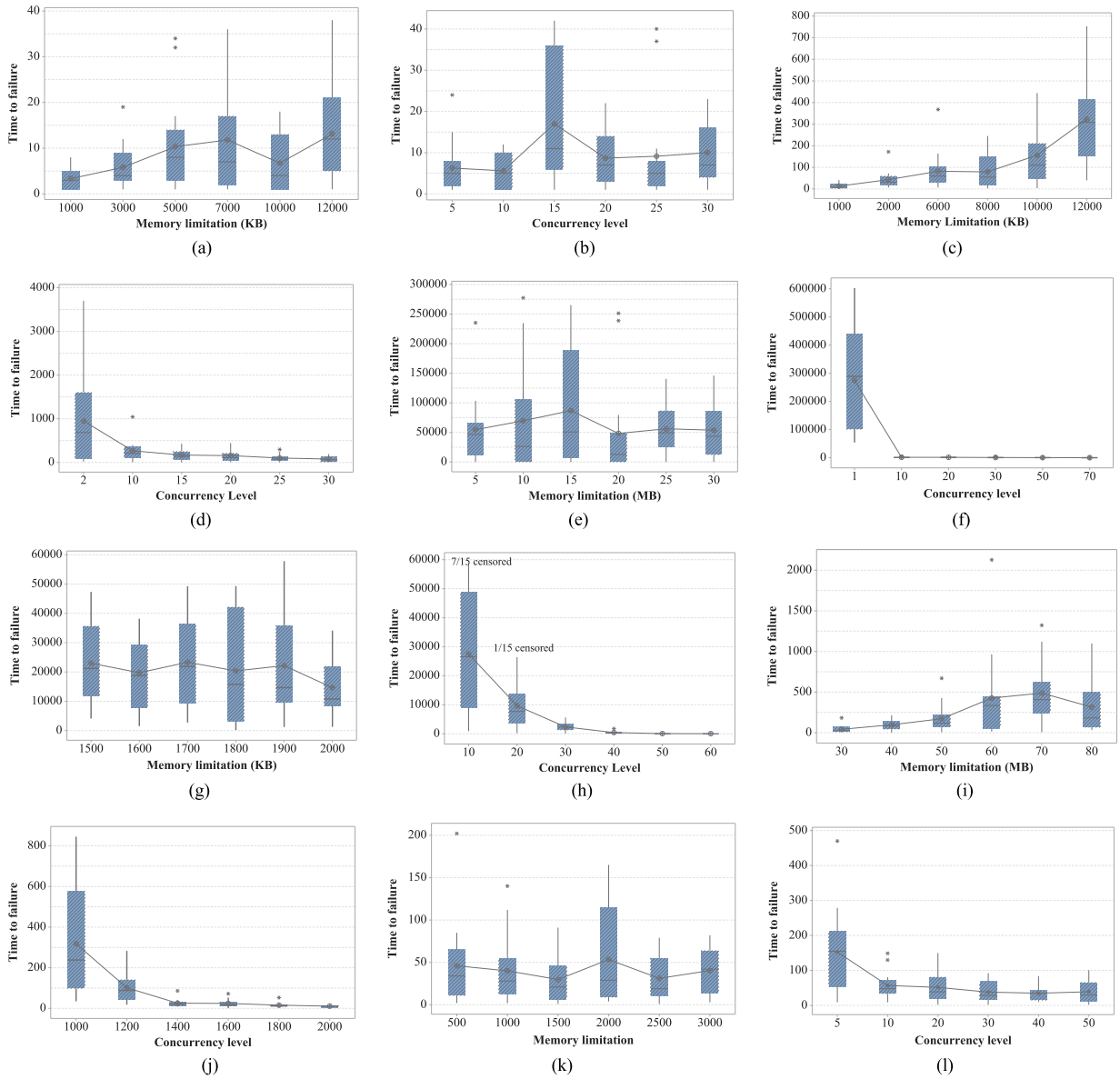

Fig. 3. Sample values of the response variable (or TTF) by influencing factors for six applications. (a) *Airline* by memory limitation. (b) *Airline* by concurrency level. (c) *Account* by memory limitation. (d) *Account* by concurrency level. (e) *MySQL* by memory limitation. (f) *MySQL* by concurrency level. (g) *Mozilla* by memory limitation. (h) *Mozilla* by concurrency level. (i) *Pbzip* by memory limitation. (j) *Pbzip* by concurrency level. (k) *Memcached* by memory limitation. (l) *Memcached* by concurrency level.

the MTTF and TTF regression models and further formalize the relationship model for the purpose of estimation.

B. Answer for RQ2

For this research question, we tested four types of relationship models, i.e., *Linear model, Exponential model, Logarithmic model, and Power model*; the mathematical models are shown in Section III-B3. The data analyzed were the eight groups of response variables from the second stage shown in Fig. 3. We adopted the least-squares regression to estimate model parameters and adopted the *s-R*-squared and *s-R*-squared adjust as the GoF indicators [40]. For the *s-R*-squared and *s-R*-squared adjust, larger values indicate better fit.

Table VI summarizes the GoF statistics and the fitting results. The second column shows the statistically significant influencing factor for each application. The third to tenth columns show the *s-R*-squared and *s-R*-squared adjust statistics for four types of relationship models. The last column presents the best-fitting MTTF regression models with respect to the explanatory variables, where ML denotes maximum limitation variable and CL denotes concurrency level variable.

Finding #4: Power model best fits the relationship between MTTF and the influencing factor explanatory variable.

According to Table VI, the *Power model* provides either the largest or the second largest values of *s-R*-squared and *s-R*-squared adjust statistics for the eight groups, and the GoF values are all greater than 80% except for that of *Airline* application,

TABLE VI
MTTF REGRESSION ANALYSIS RESULTS

App.	Inf.	*Linear model*		*Exponential model*		*Logarithmic model*		*Power model*		Fittest Regression Model
		s-R-sq	s-R-sq (adj)	s-R-sq	s-R-sq (adj)	s-R-sq	s-R-sq (adj)	s-R-sq	s-R-sq (adj)	
Airline	M.L.	49.5%	36.8%	52.0%	40.0%	60.8%	51.0%	**71.1%**	**63.8%**	$MTTF_{Airline}(ML) = e^{-2.015} * ML^{0.4782}$
Account	M. L.	75.9%	69.9%	**89.7%**	**87.1%**	56.5%	45.6%	**89.3**%	**86.6**%	$MTTF_{Account}(ML) = e^{-4.512} * ML^{1.044}$
	C. L.	70.4%	62.9%	**92.3%**	**90.4%**	95.7%	94.6%	**98.1**%	**97.7**%	$MTTF_{Account}(CL) = e^{-7.536} * CL^{-0.8878}$
MySQL	C. L.	31.0%	13.7%	76.3%	70.4%	72.6%	65.8%	**95.5%**	**94.4%**	$MTTF_{MySQL}(CL) = e^{12.30} * CL^{-2.075}$
Mozilla	C. L.	78.5%	73.1%	**97.5%**	**96.8%**	93.8%	92.2%	87.1%	83.8%	$MTTF_{Mozilla}(CL) = e^{11.62-0.1414*CL}$
Pbzip	M. L.	67.7%	59.7%	78.8%	73.5%	72.3%	65.3%	**87.1%**	**83.9%**	$MTTF_{Pbzip}(ML) = e^{-4.348} * ML^{2.426}$
	C. L.	63.8%	54.8%	88.6%	85.8%	72.6%	65.8%	**93.5%**	**91.8%**	$MTTF_{Pbzip}(CL) = e^{38.49} * CL^{-4.780}$
Memcached	C. L.	50.2%	37.7%	62.3%	52.8%	74.5%	68.2%	**84.5%**	**80.6%**	$MTTF_{Memcached}(CL) = e^{5.683} * CL^{-0.5705}$

indicating the accuracy of the utilized regression models. By contrast, the best-fitting model for *Account* and *Mozilla* is *Exponential model*, and their corresponding GoF values are all greater than 87.1%. The values corresponding to the *Power model* are all greater that 80%, indicating that the *Power model* can also be used to describe the relationship. In the following, we show the regression lines along with the response variable values with respect to the memory limitation settings and concurrency level settings.

1) By Memory Limitation Settings: Figs. 4(a) and (d) and 5(a) show the regression lines of the logarithmically transformed MTTF, i.e., MTTF (transformed), with respect to the logarithmically transformed memory limitation variable, i.e., ML (transformed), for *Airline*, *Account*, and *Pbzip*, respectively. In the subfigures, the squared dot denotes the transformed MTTF, which is calculated by (1), the line denotes the regression line, whose mathematical formula is shown in the last column of Table VI, and the natural logarithmic coordinates are used for better illustration. In the subfigures, we can observe that the dots scatter around the best-fitting lines, indicating the high accuracy of the regression analysis and the sufficiency of the utilized candidate models.

On the other hand, for the trends of the MTTF variations, we can observe that the regression lines for memory limitation factor all have positive slope magnitudes, i.e., 0.4782, 1.044, and 2.426, confirming that the expected time to data race failures can be reduced by decreasing the memory limitation settings. In addition, the slope magnitudes differ among the applications, indicating that some applications are more sensitive to the memory limitation settings. In our opinion, the differences are caused by the different memory usage intensities among applications. For example, although *Pbzip* and *Memcached* have almost the same number of lines of instructions, *Pbzip* requires much more memory to run, i.e., approximately 37 360 KB compared to 3498 KB for *Memcached* (see Table II). Thus, according to this observation, we can have an implication that memory limitation is more likely to accelerate the applications that have a larger memory usage intensity.

2) By Concurrency Level Settings: Figs. 4(g) and (j) and 5(d) and (g) illustrate the regression lines of the logarithmically transformed MTTF, i.e., MTTF (transformed), with respect to logarithmically transformed concurrency level variable, i.e., CL (transformed), for *Account*, *MySQL*, *Pbzip*, and *Memcached*, respectively. Similarly, the regression lines' mathematical formulas are shown in the last column of Table VI. Fig. 4(m) shows the regression line of transformed MTTF with respect to concurrency level variable for *Mozilla* application. For these five subfigures, all the dots scatter around the regression lines, indicating the tightness of the regression analysis.

The trends of the MTTF variations are opposite to those for memory limitation settings, and the regression lines have negative slope magnitudes of -0.8878, -2.075, -0.1414, -4.780, and -0.5705, respectively. These slope values support the observation that the expected time to data race failure can be reduced by increasing the concurrency level.

C. Answer for RQ3

For this research question, we explored the relationship model between TTF and the influencing explanatory variables, i.e., memory limitation and the concurrency level, by maximum-likelihood estimation [20]. Two steps are included in maximum-likelihood estimation to make the exploration. First, a probability distribution with unknown parameters is constructed for the collected data. Second, those unknown parameters are estimated by maximizing the samples' (log-) likelihood function, which requires the assumed distribution's *pdf* and *cdf*. Compared to the least-squares estimation, maximum-likelihood estimation can handle censored observations and Non-Normal assumed distributions, such as *Weibull* distribution. The assumed distribution model for a TTF random variable is constructed by combining the relationship model and the local distribution model into an integrated model. Since we use influencing factor S as the explanatory variable for the regression model, the TTF can be expressed in a general form of (2).

$$t(s) = \mu(s) + \sigma \cdot \epsilon \tag{2}$$

where $\mu(s)$ denotes the s-expectation when $S = s$ and is also known as the relationship model between the influencing factor and the response variable (or TTF), σ denotes the scale parameter, and ϵ denotes the random item. From this equation, we known that ϵ determines the randomness of TTF when the influencing factor is set to be a certain value s and also determines TTF's distribution model. The distribution for ϵ is known as the

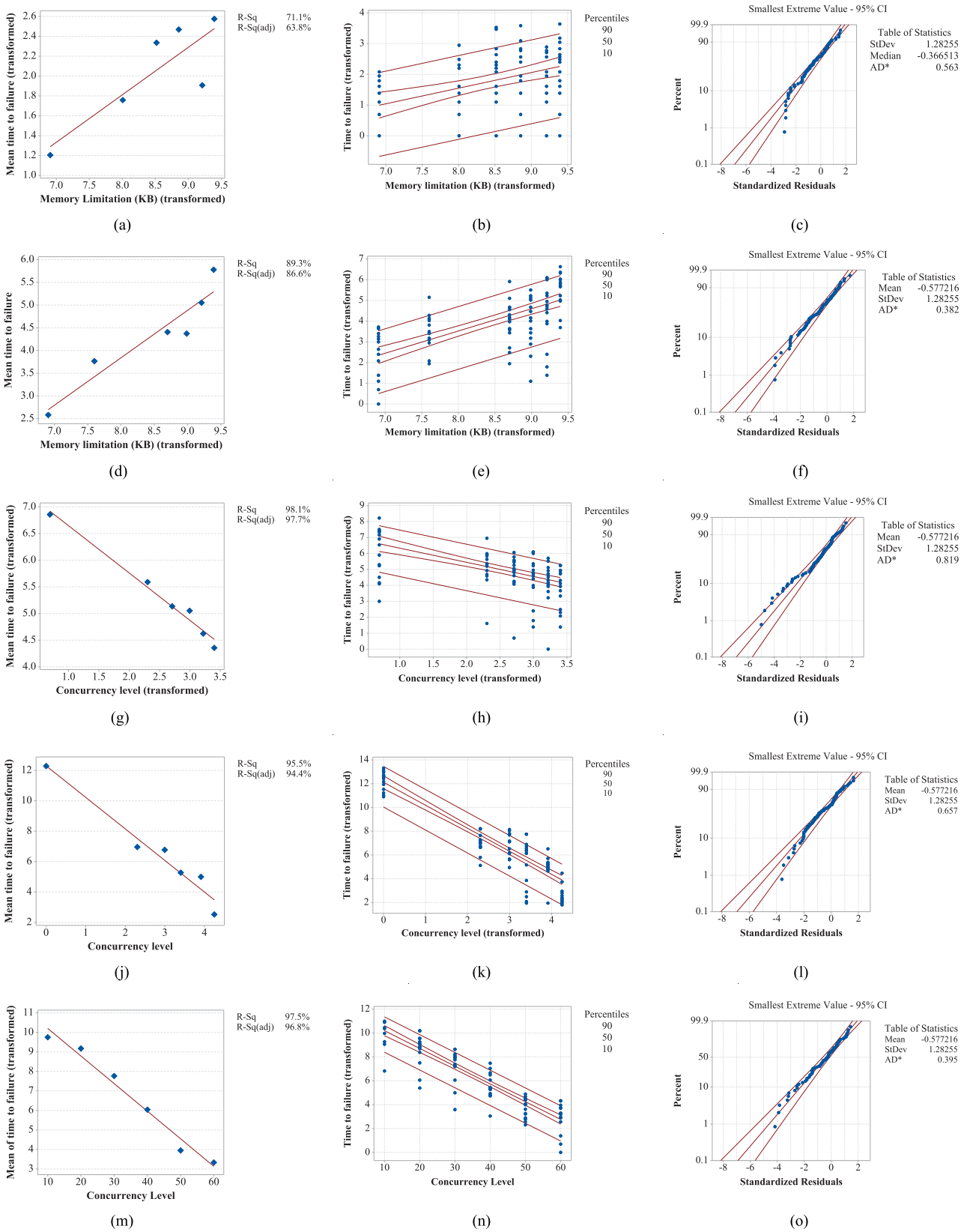

Fig. 4. TTF/MTTF regression results by influencing factors for six applications (part 1). (a) *Airline* MTTF regression by ML. (b) *Airline* TTF regression by ML. (c) *Airline* TTF *s*-standardized residuals plot by ML. (d) *Account* MTTF regression by ML. (e) *Account* TTF regression by ML. (f) *Account* TTF *s*-standardized residuals plot by ML. (g) *Account* MTTF regression by CL. (h) *Account* TTF regression by CL. (i) *Account* TTF *s*-standardized residuals plot by CL. (j) *MySQL* MTTF regression by CL. (k) *MySQL* TTF regression by CL. (l) *MySQL* TTF *s*-standardized residuals plot by CL. (m) *Mozilla* MTTF regression by CL. (n) *Mozilla* TTF regression by CL. (o) *Mozilla* TTF *s*-standardized residuals plot by CL.

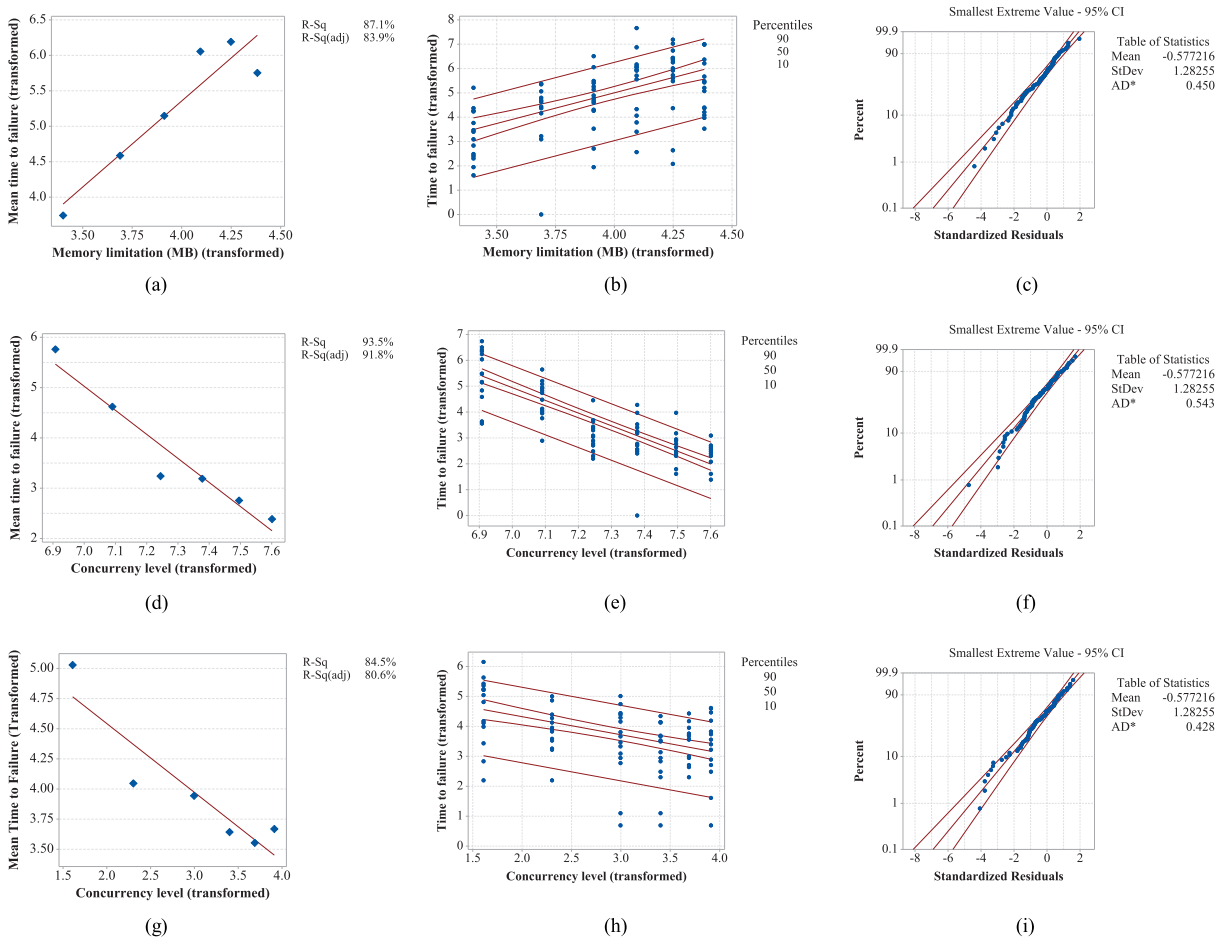

Fig. 5. TTF/MTTF regression results by influencing factors for six applications (part 2). (a) *Pbzip* MTTF regression by ML. (b) *Pbzip* TTF regression by ML. (c) *Pbzip* TTF *s*-standardized residuals plot by ML. (d) *Pbzip* MTTF regression by CL. (e) *Pbzip* TTF regression by CL. (f) *Pbzip* TTF *s*-standardized residuals plot by CL. (g) *Memcached* MTTF regression by CL. (h) *Memcached* TTF regression by CL. (i) *Memcached* TTF *s*-standardized residuals plot by CL.

local distribution, and TTF's distribution is known as the overall distribution model.

Based on the data we collected in the second stage experimental data, we first figure out the local distribution. Then based on (2), we second construct the overall model associated with relationship candidate models. With the help of maximum-likelihood estimation, we calculate the unknown parameters and GoF statistics and determine the best-fitting model among the candidates. In the following, we will divide the analyses and findings into two parts according to the steps of the analysis procedure as presented above for each application.

Step 1: Local distribution analysis

According to the analysis conducted in Section IV-A, there were eight groups (six settings for each) of sensitive data as shown in Fig. 3. Therefore, a total of 48 sets of responsible values under different settings were used. In this part, we tested four candidate distribution models: *Weibull* distribution, *Log-Normal* distribution, *Exponential* distribution, and *Normal* distribution, whose *pdf* and *s*-mean (or MTTF) formulas are shown in (3)–(10). We applied the Anderson–Darling (AD) adjusted [41] statistic as the GoF. For the AD statistic, values closer to zero indicate better fit.

1) *Weibull distribution*

$$f(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} e^{-\left(\frac{t}{\eta}\right)^{\beta}} \tag{3}$$

$$\text{MTTF} = \eta \cdot \Gamma\left(1 + \frac{1}{\beta}\right) \tag{4}$$

where β denotes the shape parameter, η denotes the scale parameter, and $\Gamma(\cdot)$ denotes the gamma function.

2) *LogNormal distribution*

$$f(t) = \frac{1}{\sqrt{2\pi}\sigma t} e^{-\frac{1}{2}\left(\frac{\log(t)-\mu}{\sigma}\right)^2} \tag{5}$$

$$\text{MTTF} = \exp\left(\mu + \frac{1}{2}\sigma^2\right) \tag{6}$$

where μ denotes the location parameter, σ denotes the scale parameter, and $\exp(\cdot)$ denotes the exponential function.

3) *Exponential distribution*

$$f(t) = \lambda e^{-\lambda t} \tag{7}$$

$$\text{MTTF} = \frac{1}{\lambda} \tag{8}$$

TABLE VII
LOCAL DISTRIBUTION ANALYSIS RESULTS

App.	Inf.	Level	*Weibull*	*LogNormal*	*Exponential*	*Normal*
Airline	M. L.	L1	**1.357**	1.507	1.631	**1.471**
		L2	**1.312**	**1.115**	1.685	1.929
		L3	**1.080**	1.186	**1.076**	1.855
		L4	**1.098**	1.209	**1.117**	1.672
		L5	**1.287**	1.393	**1.255**	1.717
		L6	**0.977**	**1.177**	1.343	1.200
Account	M. L.	L1	**1.078**	**1.116**	1.171	1.718
		L2	**1.150**	**1.164**	1.335	1.907
		L3	**1.105**	**1.051**	1.150	2.166
		L4	**1.027**	1.189	**1.054**	1.496
		L5	**1.142**	1.404	**1.170**	1.801
		L6	**1.068**	1.377	1.928	**1.100**
	C. L.	L1	**1.253**	**1.245**	1.561	1.607
		L2	**1.298**	1.612	**1.562**	1.988
		L3	**1.145**	1.889	1.746	**1.159**
		L4	**1.142**	1.404	**1.170**	1.801
		L5	**1.145**	1.717	**1.427**	1.564
		L6	**1.198**	1.354	**1.183**	1.335
MySQL	C. L.	L1	**1.121**	1.340	1.780	**1.107**
		L2	**1.402**	**1.298**	1.946	2.037
		L3	**1.256**	1.283	**1.246**	1.548
		L4	**1.440**	**1.678**	2.578	2.062
		L5	**2.041**	**2.358**	2.793	2.549
		L6	**2.414**	**1.997**	2.428	3.483
Mozilla	C. L.	L1	**1.780**	1.954	1.848	**1.782**
		L2	**1.254**	1.66	**1.289**	1.553
		L3	**1.300**	1.818	1.458	**1.204**
		L4	**1.456**	**1.285**	1.457	2.384
		L5	**1.195**	**1.189**	1.46	1.463
		L6	**1.223**	1.532	**1.221**	1.358
Pbzip	M. L.	L1	1.197	**1.103**	**1.193**	2.057
		L2	**1.423**	2.016	1.740	**1.295**
		L3	**1.045**	1.192	**1.077**	1.886
		L4	**1.281**	**1.389**	1.446	2.373
		L5	**1.300**	1.886	1.370	**1.197**
		L6	**1.269**	**1.159**	1.293	2.247
	C. L.	L1	**1.187**	1.303	**1.193**	1.443
		L2	**1.058**	**1.157**	1.593	1.281
		L3	**1.291**	**1.042**	2.051	1.878
		L4	**1.284**	**1.511**	1.846	1.693
		L5	**1.536**	**1.205**	2.441	2.149
		L6	**1.212**	1.449	2.984	**1.206**
Mem-cached	C. L.	L1	**1.164**	1.367	**1.296**	1.305
		L2	**1.253**	**1.104**	2.159	1.709
		L3	**1.036**	1.362	**1.138**	1.290
		L4	**1.065**	**1.193**	1.365	1.328
		L5	**1.180**	**1.258**	2.231	1.335
		L6	**1.038**	1.149	**1.054**	1.503

where λ denotes the rate parameter.

4) *Normal distribution*

$$f(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} \tag{9}$$

$$\text{MTTF} = \mu \tag{10}$$

where μ denotes the location parameter and σ denotes the scale parameter.

Table VII summarizes the AD statistics of the four distributions for the 48 sets of responsible values under different settings. We highlighted the smallest and the second smallest values with bold font.

Finding #5: Weibull distribution model best fits for the TTF distribution model of data race failures.

According to Table VII, we observe that the column representing *Weibull* distribution presents the largest proportion (47 of 48 settings) of the smallest or second smallest AD statistics, followed by *LogNormal* (22 of 48), *Exponential* (18 of 48), and *Normal* (9 of 48) distribution columns. In addition, from the perspective of a single application, *Weibull* distribution also sees the largest proportion.

Steps 2 and 3: Overall distribution model analysis and MTTF calculation

For the maximum-likelihood estimation, we first constructed the mathematical forms of the overall distribution models by combining the local distribution model with the relationship models, according to the overall model integrating algorithms described in the literature [20]. Because the fittest distribution for the applications was *Weibull* distribution, we obtained two types of candidate overall models, i.e., *Exponential Weibull distribution model* and *Power Weibull distribution model*, whose *pdfs* are shown by (11) and (12), respectively. We applied the *s*-standardized residuals plot [20] as the GoF indicator for each group of samples.

Then, for the overall model candidates, the maximum-likelihood estimation was used to estimate the unknown parameters. The *s*-residuals are the difference values between the prediction values and the observations. After standardization according to [20], the *s*-standardized residuals should look like the samples of random variable ϵ. For example, when the local distribution is *Weibull* distribution, whose logarithmic transformation random variable is *Smallest Extreme Value* distribution [20], the expected *s*-standardized residuals should look like the samples of the standardized *Smallest Extreme Value* random variable. Then, if the *s*-standardized residuals cluster around the probability function line and fall in 95% confidence interval (CI) lines, we say the regression model is adequate.

1) *Exponential Weibull distribution model*

$$f(t; S=s) = \frac{1}{\sigma\beta_0 e^{\beta_1 s}} \left(\frac{t}{\beta_0 e^{\beta_1 s}}\right)^{\frac{1}{\sigma}-1} e^{-\left(\frac{t}{\beta_0 e^{\beta_1 s}}\right)^{\frac{1}{\sigma}}} \tag{11}$$

which is obtained by setting β as $1/\sigma$ and setting η as $\beta_0 e^{\beta_1 S}$ (*Exponential model*) for (3).

2) *Power Weibull distribution model*

$$f(t; S=s) = \frac{1}{\sigma\beta_0 s^{\beta_1}} \left(\frac{t}{\beta_0 s^{\beta_1}}\right)^{\frac{1}{\sigma}-1} e^{-\left(\frac{t}{\beta_0 s^{\beta_1}}\right)^{\frac{1}{\sigma}}} \tag{12}$$

which is obtained by setting β as $1/\sigma$ and setting η as $\beta_0 S^{\beta_1}$ (*Power model*) for (3).

Where s denotes a value of S influencing factor variable, which is a value of memory limitation setting or concurrency level setting, and t denotes a value of the TTF. By comparing the fitting results of overall model candidates, we found the following two findings.

Finding #6: Power Weibull distribution model best fits the relationship between TTF and the influencing factor explanatory variable.

TABLE VIII
TTF REGRESSION ANALYSIS RESULTS

App.	Inf.	Param.	Estimator	Std. Error	95% *Normal* CI	
					Lower-B	Upper-B
Airline	M. L.	$log(\beta_0)$	-2.18789	1.000340	-4.14852	-0.22726
		β_1	0.507753	0.117541	0.277377	0.738128
		σ	0.884032	0.071554	0.754348	1.03601
Account	M. L.	$log(\beta_0)$	-4.69646	0.910114	-6.48025	-2.91267
		β_1	1.07249	0.107280	0.862226	1.28276
		σ	0.980889	0.080328	0.835435	1.15167
	C. L.	$log(\beta_0)$	7.56417	0.310450	6.95570	8.17264
		β_1	-0.88832	0.113976	-1.11171	-0.66494
		σ	0.944139	0.080271	0.79922	1.11533
MySQL	C. L.	$log(\beta_0)$	12.5242	0.27342	11.9883	13.0601
		β_1	-1.93619	0.08828	-2.10922	-1.76317
		σ	1.10709	0.090199	0.943702	1.29878
Mozilla	C. L.	$log(\beta_0)$	12.0311	0.267986	11.5059	12.5564
		β_1	-0.14866	0.006562	-0.16152	-0.1358
		σ	1.04336	0.091513	0.878563	1.23906
Pbzip	M. L.	$log(\beta_0)$	-4.69499	1.46117	-7.55883	-1.83115
		β_1	2.51977	0.367641	1.7992	3.24033
		σ	1.04330	0.086479	0.88686	1.22734
	C. L.	$log(\beta_0)$	39.6112	2.07065	35.5528	43.6695
		β_1	-4.91551	0.283713	-5.47157	-4.35944
		σ	0.707349	0.057116	0.603813	0.828639
Mem-cached	C. L.	$log(\beta_0)$	5.83177	0.307033	5.23000	6.43354
		β_1	-0.60390	0.098259	-0.796480	-0.411310
		σ	0.817677	0.067593	0.695372	0.961494

Finding #7: The MTTF and TTF can be accurately estimated by using the proposed influencing factors as explanatory variables.

In the following, we present the details of the analysis. Table VIII summarizes the regression results of the best-fitting overall model, where β_0, β_1, and σ are the model parameters of the overall distribution *pdfs* described in (11) and (12).

According to Table VII, *Weibull* distribution was the best-fitting distribution for *Airline*, *Account*, *MySQL*, *Pbzip*, and *Memcached* applications. We found that the *Power Weibull distribution* had better fitness than that of the *Exponential Weibull distribution* for corresponding values of the response variable analyzed. The estimated model parameters, in terms of β_0, β_1, and σ, and their *s*-confidence intervals (CI) are shown in Table VIII.

Figs. 4(b) and (e) and 5(b) show the logarithmically transformed TTF's 90th, 50th, and 10th percentile lines with respect to the logarithmically transformed memory limitation variable for the *Airline*, *Account*, and *Pbzip* applications, where the dots denote the values of the response variable collected from the second stage for memory limitation factor. In these subfigures, most of the dots fall within the 90th and 10th percentile lines, indicating the high accuracy of the regression analysis. Moreover, there are clear upward trends and different magnitudes of slope among applications, which supports the findings in Section IV-B.

Figs. 4(c) and (f) and 5(b) compare the *s*-standardized residuals plot of the memory limitation experiments with the standardized *Smallest Extreme Value* distribution for *Airline*, *Account*, and *Pbzip* applications, respectively. In these figures, the red lines denote the probability function line and its 95% CI for standardized *Smallest Extreme Value* distribution, and the blue points denote the calculated *s*-standardized residuals. According to the subfigures, we can observe that most *s*-standardized residuals fall into the 95% CI lines, indicating the good accuracy of the *Power Weibull* distribution model.

The MTTFs for *Airline*, *Account*, and *Pbzip* influenced by ML, memory limitation variable, can be calculated according to (4), and the specific forms are given by the following equations:

$$\text{MTTF}_{Airline}(\text{ML}) = 0.956395 \times e^{-2.18789} \times \text{ML}^{0.507753} \tag{13}$$

$$\text{MTTF}_{Account}(\text{ML}) = 0.992070 \times e^{-4.69646} \times \text{ML}^{1.07249} \tag{14}$$

$$\text{MTTF}_{Pbzip}(\text{ML}) = 1.019086 \times e^{-4.69499} \times \text{ML}^{2.51977}. \tag{15}$$

Figs. 4(h) and (k) and 5(e) and (h) show the logarithmically transformed TTF's 90th, 50th, and 10th percentile lines by the logarithmically transformed concurrency level variable for the *Power Weibull distribution*, along with collected response values. It can be observed that most of the dots fall within the 90th and 10th percentile lines, indicating the high accuracy of the regression analysis. In contrast to the case for memory limitation, the percentile lines show obvious downward trends.

Figs. 4(i) and (l) and 5(e) and (h) show the *s*-standardized residuals plot with respect to the standardized *Smallest Extreme Value* distribution for *Account*, *MySQL*, *Pbzip*, and *Memcached* applications. Most of *s*-standardized residuals scattered within the 95% CI lines, indicating the good accuracy of the overall distribution model.

In addition, the MTTFs for *Account*, *MySQL*, *Pbzip*, and *Memcached* by CL, concurrency level variable, can be calculated according to (4), and their specific forms are given as follows:

$$\text{MTTF}_{Account}(\text{CL}) = 0.977654 \times e^{7.56417} \times \text{CL}^{-0.88832} \tag{16}$$

$$\text{MTTF}_{MySQL}(\text{CL}) = 1.050109 \times e^{12.5242} \times \text{CL}^{-1.93619} \tag{17}$$

$$\text{MTTF}_{Pbzip}(\text{CL}) = 0.910052 \times e^{39.6112} \times \text{CL}^{-4.91551} \tag{18}$$

$$\text{MTTF}_{Memcached}(\text{CL}) = 0.936195 \times e^{5.83177}\text{CL}^{-0.60390}. \tag{19}$$

For *Mozilla* application, the overall distribution model is *Exponential Weibull distribution* (11). According to Table VII, the best-fitting distribution for concurrency level factor is *Weibull* distribution, whose *pdf* is given by (3). Compared with *Power Weibull distribution model*, *Exponential Weibull distribution model* fits the values of the response well. The model's parameters, in terms of β_0, β_1, and σ, and their CIs are shown in Table VIII. Fig. 4(n) shows the logarithmically transformed TTF's 90th, 50th, and 10th percentile lines with respect to the transformed concurrency level settings for *Exponential Weibull distribution*. The subfigure shows that most of the dots fall within the 90% and 10% percentile lines. Fig. 4(o)

shows the *s*-standardized residuals plot with respect to the standardized *Smallest Extreme Value* distribution. Since all of the *s*-standardized residuals fall into the 95% CI lines, we can conclude that *Exponential Weibull* is accurate.

The MTTF for *Mozilla* by its CL, concurrency level variable, is obtained by (4), and is given as follows:

$$\text{MTTF}_{Mozilla}(\text{CL}) = 1.019113 \times e^{12.0311-0.14866\times \text{CL}}. \quad (20)$$

Discussion: What are the relations between the MTTF formulas obtained in Section IV-B, and those in this section? Two types of MTTF formulas are obtained in this paper. In Section IV-B, the MTTF formula (see Table VI) is estimated with the mean values of the response variable, which require a large number of samples to support the correctness of the statistical result considering the central limit theorem's prerequisite conditions. Comparatively, in Section IV-C, the MTTF formula is calculated based on the TTF's *pdf*, which requires the prior knowledge of the TTF's distribution model. Although the MTTF formulas are obtained in two ways, we have the following two observations. First, the corresponding applications' MTTF mathematical forms are all the same. For example, for *Airline* application, its MTTFs are both in the form of *Power model*. Second, the differences between the corresponding intercept parameters and the corresponding slope parameters for the six applications are quite negligible.

On the basis of above two observations, we obtain the following *implication*. If the number of samples is sufficiently large, the MTTF can be estimated by the mean values of the response variable as shown in Section IV-B, which is easier to calculate. If the samples are hard to be collected to support a statistical analysis, the MTTF could be more accurately estimated by taking advantage of the prior knowledge of the TTF's distribution model, which is a *Weibull* distribution as presented in *Finding #4*.

V. Threats to Validity

In this section, we will discuss the threats to validity from two aspects.

A. External Validity

Although we perform our empirical studies on six different applications suffering from real data race failures, we still cannot claim that the effectiveness of our findings can be generalized to all software. However, to ensure the generalizability, we select the applications from four different repositories, namely *SIR* [34], *Radbench* [36], *SCTbenchmark* [37], and *MySQL* bug website [35]. The applications cover different working scenarios, e.g., desk applications (*Pbzip* and *Mozilla*), server programs (*MySQL* and *Memcached*), and business software (*Airline* and *Account*). Additionally, our applications cover three common programming languages, C (*Memcached*), C++ (*Pbzip*, *MySQL*, and *Mozilla*), and Java (*Airline* and *Account*). Considering these three languages have different primitive mechanisms to accomplish synchronization for concurrent threads, we believe that the findings obtained from the selected applications can be applied extensively for a wide range of situations.

The applications we tested are deployed on a single machine; thereby we cannot claim that all findings are suitable for applications deployed on HPC systems [28], such as on clouds. The HPC applications deal with huge sets of data and complex algorithm on hundreds of computers. However, the concurrency level and parallel level should also be used for the accelerating factors for HPC applications. Because high number of clients should increase the complexity of scheduling results; High number of deployed machines should increase the probability of modifying the unprotected shared resources simultaneously. Therefore, taking into account the synchronization mechanism similarities between the desktop/server and HPC applications, we may be able to generalize part of our findings, for example, *Finding #1, #2, #3, and #5*, to HPC applications.

B. Internal Validity

Since the time to a data race failure relies strongly on the schedulers' scheduling results, which can be impacted by other running processes spawned by the OS or other concurrent running applications, the current testing environments are another threat. To alleviate this problem, we perform three operations in our experiments to minimize the influences. First, we manually reboot the machine after the samples are collected from different stress settings. Second, we add a break time between two adjacent workload iterations under the same stress setting. Because a process's scheduling priority can be affected by its preceding ones in the queue, we use the time break to minimize this impact. Third, we will not perform any other unnecessary tasks on the testing system during the experiment.

Using the virtual machines could contribute to another threat. The side effects of using a virtual machine are the various unknown configurations different from real machines. For example, compared to a virtual machine, a real machine always has faster I/O bandwidth for the same files and the same websites. Those hidden differences could affect the scheduling results and generate different TTF observations. However, such unknown configurations cannot affect the influencing factors and their relationship with TTF/MTTF. The influencing factors are selected based on the motivation that they have physical impacts on OS schedulers. The relationship model is a quantitative description of those impacts. The impacts, which are presented in Section II, are independent of hardware machines. Therefore, our findings can be applied to both virtual machine and real machine scenarios. In addition, virtual machines are commonly used as experimental platforms when studying reliability characteristics, such as failure rate, for software failures caused by data race bugs in previous studies, such as in [26] and [37].

VI. Related Work

In this section, we review and associate our findings with related work.

A. Data Race Fault Mitigation Techniques

Various approaches have been proposed to improve the efficiency and effectiveness of testing for flushing out data race bugs. The studies [5]–[7], [42] concentrate on detecting data

race bugs by explicitly controlling the OS's scheduler, aiming at covering all the possible schedule possibilities. A serious limitation, despite considerable effort spent to overcome, is that they are not suitable for large-scale systems because of their complexity explosion and time overhead issues. The approach of using stress testing by controlling programs' external environmental conditions is inspired by research works [26], [43], and [44], in which the environmental-dependent bugs are discussed. Besides, the approach employed in this paper, which is controlling programs' environmental conditions, can effectively alleviate the complexity and time overhead issues for stress testing methods, compared with controlling OS's scheduler. In this paper, *Finding #1* confirms the effectiveness of using environmental factors to increase the occurrence of data races. *Finding #2* presents an optimal combination of environmental conditions, which can improve the efficiency of debugging and reproducing race conditions.

However, even with these efforts, it is still necessary to deal with potential failures caused by data race bugs, in particular for a safety-critical system. Fault mitigation techniques are employed to handle the potential failures and improve systems' dependability [11], [45]–[47]. A system always consists of different components. When component failures are detected by the failure management part, the fault mitigation mechanism should keep the failures from contributing to the hazard of the whole system. Since different types of bugs can cause various failure behaviors, fault mitigation methods are therefore different. Based on their major failure behaviors, bugs can be classified into bohrbugs and mandelbugs. Bohrbugs cause easy-to-reproduce and deterministic software failures, whereas mandelbugs cause hard-to-reproduce and nondeterministic failures. To mitigate the potential failures caused by bohrbugs, *N* versions or data diversity is adopted. However, due to the nondeterministic failure behavior caused by mandelbugs, recovery strategies can be used to eliminate the failures within a short period [11], [12].

Recovery strategies take advantage of diversifying the failure components' working environment conditions, such as Restart, Reboot, and Reconfiguration actions [12]. Those recovery methods, also known as the *environmental diversity* methods [11], [12], can change the environmental conditions and affect the activation process of data race bugs. Recovery approaches can increase the availability for the whole system for two reasons. First, without having to stop the machine to execute debugging and repairing, the recovery actions can be accomplished in a short time and let the system return to normal. Second, after recovery and changing environmental conditions, the normal operation time (or TTF/MTTF) is expected to be enlarged.

Failures caused by data race bugs, as typical examples of mandelbugs, are expected to be alleviated with recovery strategies. For example, when race conditions are detected by fault management part [46], reconfiguring applications' environmental conditions and retrying the failed inputs/workloads should eliminate the failures. However, there is no formal evidence to verify its feasibility. In this paper, *Finding #3* provides the statistical evidence for the *environmental diversity* methods because the TTF/MTTF varies with different environmental conditions.

B. Stochastic Process Models and TTF/MTTF Estimation

For a given system, it is necessary to conduct quantitative analysis to access the dependability metrics, such as safety, reliability, and availability by building mathematical models. The models should consider all the failures and fault mitigation behaviors. Stochastic process models, such as CTMC, SMP, Markov regenerative process (MRGP), and Petri Net, can reflect complex system behaviors [13]. A stochastic process model includes multiple states and the associated transition time between them. Given all the time parameters of consistent components, the quantitative metrics can be calculated. For example, CTMC models are constructed for IBM WebSphere SIP Applications that are equipped with mandelbug mitigation methods.

To solve a stochastic model, the parameters of TTF/MTTF (the transition time between states) for software need to be known in the first place. However, the difficulty for failures caused by data race bugs is that they are usually manifested in a rather long time. How to accelerate testing process and to estimate TTF/MTTF for hard-to-reproduce software failures have always been a fundamental question to be solved. To the state of the art, the literature of adopting ALT/ADT [19] techniques have been proposed on solving the problem by focusing on software failures caused by ARBs, which can cause accumulation of errors and lead to system performance degradation or crash problems [10]. For example, Matias *et al.* [16], [17] first proposed and evaluated the method of using ALT/ADT to reduce samples' collection time and estimate the MTTF for a web server system suffering from ARBs. Zhao *et al.* [18] studied the effectiveness of ALT approach on estimating MTTF by injecting memory leaking bugs.

Motivated by the researches in [16]–[18], how to use ALT/ADT techniques to test data race failures is a natural research question. In this paper, we empirically study two critical issues with the techniques, i.e., the influencing factors and their relationship models with TTF/MTTF. Note that because the processes of activating and propagating data race bugs are quite different from that of memory leaking bugs [44], their influencing factors cannot be shared. To the best of our knowledge, this is the first paper focusing on using regression method to reduce the data race failures' TTF/MTTF estimation problem. We propose memory limitation, concurrency level, and parallel level as influencing factors. Then, we explore the relationship models between TTF/MTTF and the influencing factors, which are shown in *Findings #4, #5, #6, and #7*.

Given *Finding #5*, SMP, MRGP, and Petri Net [13], rather than CTMC (the *Exponential* transition time distribution assumption), should be considered as more precise and accurate stochastic models for the system under test. Given *Findings #4, #6, and #7*, when several race conditions are detected, both their failure time and current influencing factor values should be dumped. Based on the failure time data and our finding relationship model, a *Power Weibull distribution model* could be estimated. The quantitative availability function with respect to different recovery methods or influencing factors could be then calculated. Taking into account the economic costs, the system should figure out the optimal fault mitigation actions to achieve the highest reliability/availability.

VII. Conclusion

In this paper we presented a study of using the stress testing with influencing factors to accelerate the manifestation process of data race software failures. In addition, it explored how to estimate the time to data race failure or mean time to data race failure for a system. Three influencing factors, namely memory limitation, concurrency level, and parallel level, were proposed to accelerate the failure process caused by a data race bug. The statistical analysis results of experiments confirmed the validity of the factors in terms of accelerating data race software failures. Therefore, the proposed influencing factors can be used to optimize the practices of software testing/debugging for data race failures.

Furthermore, the regression analysis showed that TTF/MTTF can be accurately estimated by treating the influencing factors as explanatory variables. Moreover, the best-fitting relationship models between TTF/MTTF and the proposed influencing factors are *Power Weibull distribution model* and *Power model*. These findings can facilitate the reliability evaluation for software systems suffering from data race failures.

In the future work, we plan to systematically study the influencing factors and stress testing approaches for software failures caused by concurrency bugs, including both data race and deadlock. We will analyze the similarities and differences of factors affecting different types of failures and their relationships with TTF or MTTF.

References

[1] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2010, pp. 447–456.

[2] J.-D. Choi, K. Lee, A. Loginov, R. O'Callahan, V. Sarkar, and M. Sridharan, "Efficient and precise datarace detection for multithreaded object-oriented programs," *ACM SIGPLAN Notices*, vol. 37, no. 5, pp. 258–269, 2002.

[3] P. Fonseca, C. Li, V. Singhal, and R. Rodrigues, "A study of the internal and external effects of concurrency bugs," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2010, pp. 221–230.

[4] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: A dynamic data race detector for multithreaded programs," *ACM Trans. Comput. Syst.*, vol. 15, no. 4, pp. 391–411, 1997.

[5] S. Park, S. Lu, and Y. Zhou, "CTrigger: Exposing atomicity violation bugs from their hiding places," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 25–36, 2009.

[6] M. Musuvathi, S. Qadeer, T. Ball, M. Musuvathi, S. Qadeer, and T. Ball, "Chess: A systematic testing tool for concurrent software," Microsoft Res., Microsoft Corporation, Redmond, WA, USA, Tech. Rep. MSR-TR-2007–149, 2007.

[7] J. Yu, S. Narayanasamy, C. Pereira, and G. Pokam, "Maple: A coverage-driven testing tool for multithreaded programs," in *Proc. ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 485–502, 2012.

[8] B. Wester, D. Devecsery, P. M. Chen, J. Flinn, and S. Narayanasamy, "Parallelizing data race detection," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 27–38, 2013.

[9] J. Alonso, M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault repairs and mitigations in space mission system software," in *Proc. 43rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2013, pp. 1–8.

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

[11] K. S. Trivedi, M. Grottke, and E. Andrade, "Software fault mitigation and availability assurance techniques," *Int. J. Syst. Assurance Eng. Manage.*, vol. 1, no. 4, pp. 340–350, 2010.

[12] M. Grottke, D. S. Kim, R. Mansharamani, M. Nambiar, R. Natella, and K. S. Trivedi, "Recovery from software failures caused by mandelbugs," *IEEE Trans. Rel.*, vol. 65, no. 1, pp. 70–87, Mar. 2016.

[13] K. S. Trivedi and A. Bobbio, *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2017.

[14] G. A. Hoffmann, K. S. Trivedi, and M. Malek, "A best practice guide to resources forecasting for the apache webserver," in *Proc. IEEE 12th Pac. Rim Int. Symp. Dependable Comput.*, 2006, pp. 183–193.

[15] K. Trivedi, D. Wang, D. J. Hunt, A. Rindos, W. E. Smith, and B. Vashaw, "Availability modeling of sip protocol on IBM WebSphere," in *Proc. 14th IEEE Pac. Rim Int. Symp. Dependable Comput.*, 2008, pp. 323–330.

[16] R. Matias, Jr., K. S. Trivedi, and P. R. Maciel, "Using accelerated life tests to estimate time to software aging failure," in *Proc. IEEE 21st Int. Symp. Softw. Rel. Eng.*, 2010, pp. 211–219.

[17] R. Matias, P. A. Barbetta, K. S. Trivedi, and P. J. Freitas Filho, "Accelerated degradation tests applied to software aging experiments," *IEEE Trans. Rel.*, vol. 59, no. 1, pp. 102–114, Mar. 2010.

[18] J. Zhao, Y. Jin, K. S. Trivedi, and R. Matias, Jr., "Injecting memory leaks to accelerate software failures," in *Proc. IEEE 22nd Int. Symp. Softw. Rel. Eng.*, 2011, pp. 260–269.

[19] W. B. Nelson, *Accelerated Testing: Statistical Models, Test Plans, and Data Analysis*, vol. 344. New York, NY, USA: Wiley, 2009.

[20] W. Q. Meeker and L. A. Escobar, *Statistical Methods for Reliability Data*. New York, NY, USA: Wiley, 2014.

[21] D. C. Montgomery, *Design and Analysis of Experiments.* New York, NY, USA: Wiley, 2017.

[22] M. Kifer and S. Smolka, *Introduction to Operating System Design and Implementation: The OSP 2 Approach*. New York, NY, USA: Springer, 2007.

[23] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, vol. 151. Wisconsin, MI, USA: Arpaci-Dusseau Books, 2014.

[24] R. Bryant, O. David Richard, and O. David Richard, *Computer Systems: A Programmer's Perspective*, vol. 281. Upper Saddle River, NJ, USA: Prentice-Hal, 2003.

[25] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: A comprehensive study on real world concurrency bug characteristics," *ACM SIGPLAN Notices*, vol. 43, no. 3, pp. 329–339, 2008.

[26] D. G. Cavezza, R. Pietrantuono, J. Alonso, S. Russo, and K. S. Trivedi, "Reproducibility of environment-dependent software failures: An experience report," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, 2014, pp. 267–276.

[27] R. V. Lenth, "Quick and easy analysis of unreplicated factorials," *Technometrics*, vol. 31, no. 4, pp. 469–473, 1989.

[28] A. Goscinski, M. Brock, and P. C. Church, "High performance computing clouds," *Cloud Computing: Methodology, Systems and Applications*. Boca Raton, FL, USA: CRC Press, 2012, pp. 221–259.

[29] P. E. McKight and and J. Najab, "Kruskal–Wallis test," *Corsini Encyclopedia of Psychology*. New York, NY, USA: Wiley, 2010.

[30] A. J. Dobson and A. Barnett, *An Introduction to Generalized Linear Models*. Boca Raton, FL, USA: CRC Press, 2008.

[31] R. Corporation, "Accelerated life testing reference," 2015. [Online]. Available: http://www.synthesisplatform.net/references/Accelerated_Life_Testing_Reference.pdf

[32] W. Mendenhall, R. J. Beaver, and B. M. Beaver, *Introduction to Probability and Statistics*. Boston, MA, USA: Cengage Learning, 2012.

[33] S. Park, R. W. Vuduc, and M. J. Harrold, "Falcon: Fault localization in concurrent programs," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, 2010, vol. 1, pp. 245–254.

[34] S. Khurshid, "Software-artifact infrastructure repository," Jan. 2018. [Online]. Available: http://sir.unl.edu/content/sir.php

[35] E. Schoenfelder, "Bug #38691 segfault/abort in update ... join while flush tables with read lock," Aug. 2008. [Online]. Available: https://bugs.mysql.com/bug.php?id=38691

[36] N. Jalbert, C. Pereira, G. Pokam, and K. Sen, "Radbench: A concurrency bug benchmark suite," *HotPar*, vol. 11, pp. 2–2, 2011.

[37] P. Thomson, A. F. Donaldson, and A. Betts, "Concurrency testing using schedule bounding," *ACM SIGPLAN Notices*, vol. 49, no. 8, pp. 15–28, 2014.

[38] P. Menage, "cgroup documentation," 2004. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

[39] Dormando, "Memcached configuring server," 2016. [Online]. Available: https://github.com/memcached/memcached/wiki/ConfiguringServer

[40] P. McCullagh, "Generalized linear models," *Eur. J. Oper. Res.*, vol. 16, no. 3, pp. 285–292, 1984.
[41] T. W. Anderson and D. A. Darling, "A test of goodness of fit," *J. Amer. Statist. Assoc.*, vol. 49, no. 268, pp. 765–769, 1954.
[42] S. Burckhardt, P. Kothari, M. Musuvathi, and S. Nagarakatte, "A randomized scheduler with probabilistic guarantees of finding bugs," *ACM SIGPLAN Notices*, vol. 45, no. 3, pp. 167–178, 2010.
[43] S. Chandra and P. M. Chen, "Whither generic recovery from application faults? A fault study using open-source software," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2000, pp. 97–106.
[44] D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi, "How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation," *J. Syst. Softw.*, vol. 113, pp. 27–43, 2016.
[45] H. Mushtaq, Z. Al-Ars, and K. Bertels, "Survey of fault tolerance techniques for shared memory multicore/multiprocessor systems," in *Proc. IEEE 6th Int. Des. Test Workshop*, 2011, pp. 12–17.
[46] F. Ye and T. Kelly, "Component failure mitigation according to failure type," in *Proc. 28th Annu. Int. Comput. Softw. Appl. Conf.*, 2004, pp. 258–264.
[47] A. Löfwenmark and S. Nadjm-Tehrani, "Fault and timing analysis in critical multicore systems—A survey with an avionics perspective," *J. Syst. Archit.*, vol. 87, pp. 1–11, 2018.

Kun Qiu received the B.S. degree in automation from the Hefei University of Technology, Hefei, China, in 2013. He is currently working toward a Ph.D. degree in control science and engineering with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, China.

His research interests are in software dependability analysis and metamorphic testing.

Zheng Zheng (SM'18) received the Ph.D. degree in computer software and theory from the Chinese Academy of Science, Beijing, China, in 2006.

In 2014, he was a Research Scholar with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA. He is currently a Professor in Control Science and Engineering with Beihang University, Beijing, China. His research interests include software dependability, unmanned aerial vehicle path planning, artificial intelligence applications, and software fault localization.

Kishor S. Trivedi (LF'17) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Mumbai, Mumbai, India, in 1968, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 1972 and 1974, respectively.

He is currently the Fitzgerald Hudson Chair with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA. His research interests are in reliability, availability, performance and survivability of computer and communication systems and in software dependability.

Mr. Trivedi is a Golden Core Member of the IEEE Computer Society. He was the recipient of the IEEE Computer Society Technical Achievement Award.

Beibei Yin received the Ph.D. degree in control science and engineering from Beihang University, Beijing, China, in 2010.

She was a Research Scholar with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, in 2015. She is currently a Lecturer with Beihang University, China. Her research interests include software testing, software reliability, and software cybernetics.