Understanding the Impacts of Influencing Factors on Time to a DataRace Software Failure

Kun Qiu, Zheng Zheng*
School of Automation Science and Electrical Engineering,
Beihang University, 100191, China
Email: qiukun@buaa.edu.cn; zhengz@buaa.edu.cn

Kishor Trivedi
Department of Electrical and Computer Engineering, Duke University,
Durham, NC 27708, USA
Email: ktrivedi@duke.edu

Beibei Yin
School of Automation Science and Electrical Engineering,
Beihang University, 100191, China
Email: yinbeibei@buaa.edu.cn

Abstract—Datarace is a common problem on shared-memory parallel computers, including multicores. Due to its dependence on the thread scheduling scheme of its execution environment, the time to a datarace failure is usually very long. How to accelerate the occurrence of a datarace failure and further estimate the mean time to failure (MTTF) is an important topic to be studied. In this paper, the influencing factors for failures triggered by datarace bugs are explored and their influences on the time to datarace failure including the relationship with the MTTF are empirically studied. Experiments are conducted on real datarace suffering programs to verify the factors and their influences. Empirical results show that the influencing factors do have influences on the time to datarace failure of the subjects. They can be used to accelerate the occurrence of datarace failures and accurately estimate the MTTF.

Keywords—Datarace, failure acceleration, influencing factor, MTTF estimation.

I. INTRODUCTION

Multithreaded and distributed programs have become mainstream, but programming them remains difficult and error prone. A particular problem is datarace, i.e., situations where two or more threads accessing the same memory location in an undetermined order and at least one of these accesses is a write operation [1]. Currently, in order to accelerate the detection of the datarace bug led failures (datarace failure in short), a lot of techniques have been proposed [1], [2], [3], [4]. However, due to the overhead of the techniques, they still cannot be employed well on large complex systems [4].

For a datarace bug, race conditions appear only when multiple threads concurrently share data, thus it may lead to inconsistent program states that are extremely difficult to reproduce [1], [5], [6]. As a result, the manifestation time to a datarace failure is expected to be long. Due to the hard-to-reproduce characteristic of datarace failures, it is usually difficult to experimentally study datarace failures by collecting of sufficient failure manifestation time samples. However, most of the availability software application models are based on the assumption that the distribution of time to failure (TTF) or mean time to failure (MTTF) of each component is known [7], [8], [9]. TTF or mean time to failure (MTTF) are also important metrics used to compare the effect of different type of bugs for software systems [10], [11]. The TTF and MTTF can also reflect the reliability of a software system and can be used to compare two systems with similar functions [12].

As a result, when a system’s software components suffering from datarace failures, two issues arise:

1) How to accelerate the occurrence of a datarace failure?
2) How to estimate the mean time to a datarace failure (MTTF)?

We present an approach of adopting stress testing [13] to solve these two issues, that is to reduce the failure time due to datarace bugs, and in further to estimate the MTTF. In this work, we propose two influencing factors, memory limitation and concurrency level, and investigate their influences on time to a datarace failure through an empirical study on two IBM datarace test benchmark programs, [14], [15] and the MySQL Server software system. These experimental subjects are real applications, suffering real datarace bugs. For each of these subjects, two independent experiments are designed to verify the two corresponding influencing factors. The influences of these two factors are concluded from statistical analysis of the variation of TTF and MTTF under different experimental conditions.

Through the analysis of experimental results, we have the following main observations: (1) Both memory limitation and concurrency level have impacts on the MTTF of the subjects. And the impacts of different factors on the same program suffering a datarace bug are different. (2) Both memory limitation and concurrency level can work as controllable factors for the reducing of MTTF. Among the candidate relationship models, Power model is a good one to fit the impacts of influencing factors on the mean time to a datarace failure. (3) The fittest distribution model of TTF changes with the variation of influencing factors. (4) Based on the characteristics studied in this work, our approach can be used to accurately estimate the MTTF.

The rest of the paper is organized as follows. Section II presents the influencing factors for datarace failure. Section III describes the experimental setup for the empirical study. In Section IV, the results of the experiments are presented by answering five research questions one by one. Section V shows the threats to validity of the work. Section VI reviews related work of accelerating techniques for software failures. At last, Section VI concludes the paper.
II. INFLUENCING FACTORS FOR DATARACE

In this section, we will start to explore the way of how to influence the occurrence of a datarace failure. Recall that, a datarace happens when there are memory accesses in a program where they all:

- target the same location;
- are performed concurrently by two or more threads;
- are not reads;
- are not synchronization operations.

Thus, the main idea of influencing a datarace failure’s occurrence is to increase their probability of writing their shared location. Based on it, we select two factors as influencing factor:

1) Memory limitation, which is maximum size of physical memory the program can resident during execution.
2) Concurrency level, which is number of the concurrent visiting clients for the program.

If a thread suffers a major fault while trying to read from a virtual memory address that is not currently mapped to a RAM address, a page fault happens [16]. In this scenario, the running thread has to be preempted and suspended to the waiting state. At this time, the data required by the thread must be swapped in from the virtual memory on disk, and in this interval the operating system usually starts to run another waiting thread. Thus, if the available physical memory decreases, it will increase the switch probability of threads. As a result, if two or more threads accesses the same memory location in an undetermined order and at least one of these accesses is a write operation, the risk of datarace will be increased [16]. Therefore, we select memory limitation as the first influencing factor.

The concurrency level controls the number of kernel threads or processes on top of which the user-level threads are mapped. It is a common observation that datarace bugs are sometimes easier to manifest at high concurrency level workload [17]. High concurrency level workload increases the context switch intensity and the competition of resource. Therefore, it increases the possibility of hitting certain orders among the threads that can trigger the bug. In this work, we select the concurrency level as the second influencing factor.

We will further verify the influence of the factors by statistical analysis in Section IV. In the following, we describe details of the experimental methods to study these two influencing factors.

III. EXPERIMENTAL SETUP

In order to verify the proposed approach, we conduct an empirical study on programs suffering from datarace failures. These failures are triggered by real reported bug or simplified real bugs. In this section, we will describe the setup of the experiments, including the experimental subjects, the setting of accelerating levels, the stopping criteria, and the experimental platform.

A. Experimental subjects

Table I summarizes the subjects of our experiments. The second column presents the brief description of the programs. In the third column, we describe the workload format we used to reproduce the datarace failure. For the information shown in other columns, it will be introduced in the subsection III-B. Two of the subjects, i.e., Airline and Account, are selected from the IBM benchmark testing suite [14], [15]. The Airline provides the service for selling tickets for an Airline cooperation. The Account serves to manage accounts of a bank. Each of them contains a datarace bug. The third subject is MySQL, which is a popular database management software system. The developers and users have published many datarace bugs for MySQL.

The datarace fault in Airline is an unprotected shared boolean variable, known as StopSales, which represents whether or not the tickets are sold out. For Account, the datarace fault is an unprotected field in a class, known as PersonalAccount, which represents banks personal account management. For MySQL, the datarace failure selected is triggered by the bug #38691 reported in MySQL bug report website [18]. The datarace fault is an incomplete implementation of managing locks for multiple tables in the function known as mysql_multi_update_prepare().

For Airline and Account, considering the fact that different failures will trigger different types of exception, we use the try–exception mechanism of Java to distinguish the datarace failure from others. For MySQL, considering different failures are corresponding to different ERROR numbers and will be logged in log files, a datarace failure can be recognized by its ERROR number 2013. Note that, the failure caused by out of memory (OOM in short) is possible to occur if the virtual memory [16] is exhausted before the occurrence of the datarace failure, causing that we cannot observe the datarace failure.

For a given program suffering datarace bugs, there are two types of workloads, the normal workload and the failure triggered workload, which can be divided based on whether they can activate a datarace bug and lead to a failure. In our experiments, the program workload only contains the failure triggered workload, also denoted as failure-workload in our context. Due to the hard-to-reproduce characteristic of datarace failures, although the failure triggered workload can lead to a failure, the occurrence of a datarace failure is uncertain. Therefore, the workload is loaded repeatedly for each program. The repeated workload execution will be stopped when the failure occurs or when the maximal execution time (MET) is reached. Since the workload execution time in seconds of Airline is close to that of Account’s, we set the MET for both Airline and Account as 5,000 repeat time. Similarly, we set MET of MySQL as 10^6 due to its shorter workload execution time. The last column of Table I summarizes the MET for each subject.

B. Stress level setting

In this part, we describe the experimental plan for our test. Table I summarizes the experimental plan. The fourth column presents the influencing factor to set. The fifth and sixth columns present the minimum and maximum stress values. The seventh column summarizes the number of stress level. The
last column of Table I summarizes the MET for each subject. In the following, we describe the details of our experimental setting.

For our experiments, the stress loading scheme is constant stress (time-independent) for multiple stress levels. The proportion of replications is the same for different stress levels. There are three subjects and two influencing factors, thus totally six groups of experiments are required to be planned in our experiments. For each group, we design 6 levels, also denoted as treatments, and for each treatment, we sample a fixed number of 15 replications with the same stress configuration. And for each replication, we load the same failure-workload. The test will be stopped either the failure occurs or the MET is reached.

In the experiments to explore the impacts of memory limitations, for Airline, we adopt (10000, 3) as its failure-workload, in which the first element indicates the number of concurrent client requesting a ticket and the second element indicates the number of tickets sold. Here, we use a Linux tool provided by `last/bin/time` to find its corresponding maximum RSS and found that it is around 140MB. According to this, the levels are designed as 20MB, 40MB, 60MB, 80MB, 100MB, and 120MB, separating the maximum RSS averagely. For Account, we adopt (5,6000) as its failure-workload, in which the first element indicates the number of concurrent business account and the second element indicates the number of personal account. Its maximum RSS is around 120MB. Thus, its stress levels are designed as 20MB, 40MB, 60MB, 80MB, 100MB, and 120MB, separating the maximum RSS averagely. For MySQL, we adopt the failure-workload provided by [18], and observed its maximum RSS is around 32MB by reading `/proc/mysqld` file. Thus, we design 5MB, 10MB, 15MB, 20MB, 25MB, and 30MB as its stress levels.

In the experiments to explore the impacts of concurrency levels, a concurrency level is denoted by the number of user-level threads running concurrently. For Airline, we set its concurrency levels as 1000, 2000, 4000, 6000, 8000, and 10000. For Account, its concurrency levels are set as 1000, 2000, 3000, 4000, 5000, and 6000. For MySQL, we design its concurrency levels to be 1, 10, 20, 30, 50, and 70.

### C. Experimental platform and process

In our experiments, we used the virtual machines as our test platform, in which we installed Ubuntu 14.04 operating system. For the virtual machine, we allocated 1G RAM and 1 CPU core. Our hardware computer is configured with Inter Core i7-3770S 3.1GHz CPU, 8G RAM, and 500G disk.

The experimental process mainly contains the following steps. First, we initialize the experiment environment, including setting the value of the influencing factor, the number of replications for each program, and the MET.

Second, we control the influencing factor, i.e., memory limitation and concurrency level. For memory limitation, we apply the `cgroup` tool [19] to create the control group and execute our test program in it. The memory limitation can be set by changing the configuration file `memory.limit_in_bytes` of that group. For the concurrency level setting of Airline and Account, it can be controlled directly by a setting in the input parameters. For MySQL, its concurrency level setting is implemented by adding the concurrent clients. We adopt the number of context switch time to justify whether this can increase the concurrency level. The context switch includes voluntary context switch and nonvoluntary context switch two parts [16]. Fig. 1 (a) (b) show the accumulated voluntary context switch time and non-voluntary context switch time for a certain period of time in seconds. In this figure, the data are collected from the `/proc/mysqld_pid` when the number of concurrent clients are configured at, High, Medium, and Low, three different levels. According to this figure, the context switch times accumulate more rapidly when the concurrent clients number is larger, which means that our concurrency level control method is feasible. For example, when we design concurrency level to 10, then we set the concurrent client number to be 10. In order to guarantee that the experimental results are not influenced by the processes in other applications, we do not open other unnecessary applications during the execution of the subject programs. After the execution environment is set up, we execute the workload. If no failure occurs, the workload will be executed repeatedly until the MET is reached.

According to the setting of our experiments, we adopt the workload repeat time before the first datarace failure manifestation as the measure of the time to datarace failure of each replication, i.e., the experimental output variable. And

### TABLE I: Experimental Subjects and Setup

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Description</th>
<th>Workload Format</th>
<th>Influencing Factor</th>
<th>Min</th>
<th>Max</th>
<th>Stress Level</th>
<th>MET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airline</td>
<td>Airline tickets purchase software</td>
<td>(# of clients, # sold tickets)</td>
<td>Memory Limitation</td>
<td>20MB</td>
<td>120MB</td>
<td>6</td>
<td>5000</td>
</tr>
<tr>
<td>Account</td>
<td>Bank accounts management software</td>
<td>(# of business account, # of personal account)</td>
<td>Memory Limitation</td>
<td>20MB</td>
<td>80MB</td>
<td>6</td>
<td>5000</td>
</tr>
<tr>
<td>MySQL</td>
<td>Database management software</td>
<td>Client with ALTER operation, Client with FLUSH operation</td>
<td>Memory Limitation</td>
<td>5MB</td>
<td>30MB</td>
<td>6</td>
<td>10^6</td>
</tr>
</tbody>
</table>

![Fig. 1: Accumulated (non-)voluntary context switch time of MySQL](image)

(a) Voluntary (b) Nonvoluntary
larger repeat time indicates longer life of the experimental subject.

IV. RESULTS AND ANALYSIS

Fig. 2 presents the overview of experimental outputs. In the figures, the vertical axe denotes the mean of time to datarace failure of a subject. The left axe denotes the memory limitation value setup, the right axe denotes the concurrency level setup, and the dots are the treatments’ mean of time to failure. The sub-figures will be described in details in the subsection IV-B. For memory limitation experiments, under the given MET setup, we obtained all the outputs whose failure are triggered by datarace bug. During the Airline’s concurrency level experiments, we did not observe datarace failure when the concurrency level is less than 8000. And for the 10000 condition, there are 6 replications whose failures are triggered by OOM, and we dropped those OOM failure data. And for Account’s concurrency level experiments, with MET be 5000, replication time be 15, and stress level be 6, we totally tried 0.45 million submissions of failure-workload, however we did not observe desired datarace failure. For MySQL’s concurrency level experiments, we observed all outputs whose failures are triggered by the datarace bug. In the next, based on experimental data, we describe the analysis details.

A. Research Questions

In this empirical study, we mainly focus on the following research questions.

RQ1. Whether the influencing factor has impacts on the MTTF?
RQ2. What is the relationship model between the MTTF and the influencing factors?
RQ3. Is the distribution model of TTF changing with the variation of influencing factors?
RQ4. What is the TTF with respect to influencing factors?
RQ5. How to estimate the MTTF?

B. Answers

RQ1: Whether the influencing factor has impacts on the MTTF?

Fig. 2 presents the variance of mean of time to failure of each level, which is calculated by equation (1), with the increasing of the two factors, i.e., memory limitations and concurrency levels.

\[
\bar{t}_i = \frac{\sum_{j} t_{ij}}{n_i}
\]  

(1)

where \(\bar{t}_i\) denotes the mean of time to failure of \(i_{th}\) level (treatment) each subject program, \(n_i\) denotes the number of replications of \(i_{th}\) level, \(t_{ij}\) denotes the time to failure of \(i_{th}\) level \(j_{th}\) replication.

We use Kruskal-Wallis test [20] method to test the following hypotheses:

1) Hypothesis 1: The MTTF of each subject cannot be affected by limitation of memory, i.e., with or without memory limitation.
2) Hypothesis 2: The MTTF of each subject cannot be affected by the different levels of memory limitations.
3) Hypothesis 3: The MTTF is not changing with the variation of concurrency levels.

Table II summarizes the results of these Kruskal-Wallis tests. In Table II, the third column, N, summarizes the number of replications tested. And the fourth and fifth column, H and DF, summarize the Kruskal-Wallis test statistics and the degree of freedom respectively. The last column summarizes the p-values. For Hypothesis 1, the data we used are from
7 levels whose workloads are the same. According to our experimental process in Section III. For each Account and MySQL, there are 105 replications. For Airline, there left 99 replications since there are 6 replications whose failures are OOM failure. For Hypothesis 2, the data we used are coming from 6 levels of memory limitation experiments. For Airline, Account, and MySQL, there are 90 replications. For Hypothesis 3, the data we used are the replications coming from 6 levels of concurrency level experiments. For MySQL, there are 90 replications. For Airline and Account, due to the hard-to-reproduce characteristic of datarace failure [5], there are not enough data observed. As a result, we did not test the Hypothesis 3 for Airline and Account.

### TABLE II: Kruskal-Wallis Test Analysis Results

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Hypothesis</th>
<th>K. S. Test</th>
<th>N</th>
<th>DF</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airline</td>
<td>Hypothesis 1</td>
<td>99</td>
<td>18.67</td>
<td>1</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td></td>
<td>Hypothesis 2</td>
<td>90</td>
<td>29.63</td>
<td>5</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td></td>
<td>Hypothesis 3</td>
<td>105</td>
<td>38.21</td>
<td>1</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Account</td>
<td>Hypothesis 1</td>
<td>105</td>
<td>24.45</td>
<td>1</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td></td>
<td>Hypothesis 2</td>
<td>90</td>
<td>4.29</td>
<td>5</td>
<td>0.508</td>
</tr>
<tr>
<td></td>
<td>Hypothesis 3</td>
<td>90</td>
<td>68.54</td>
<td>5</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>MySQL</td>
<td>Hypothesis 1</td>
<td>105</td>
<td>68.54</td>
<td>1</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td></td>
<td>Hypothesis 2</td>
<td>90</td>
<td>4.29</td>
<td>5</td>
<td>0.508</td>
</tr>
<tr>
<td></td>
<td>Hypothesis 3</td>
<td>90</td>
<td>68.54</td>
<td>5</td>
<td>&lt; 0.001</td>
</tr>
</tbody>
</table>

From Table II, we can observe that, for the tests of the first hypothesis, the p-values are all less than significant level, 0.001, for the three programs. Therefore, we can conclude that there is a significant evidence that all programs’ MTTF are affected by memory limitations. For the second hypothesis, there is a significant evidence that Airline’s and Account’s MTTFs are changing with the variation of memory limitation value. However, the p-value for the test on MySQL is 0.508, which is greater than 0.001. It indicates that, although the influencing factor has significant impacts on the MTTF, MySQL’s MTTF do not change greatly with the variation of memory limitations. For the third hypothesis, we can observe that, there is a significant evidence that MySQL’s MTTF changes with the variation of concurrency levels.

Further according to Fig. 2a and 2b, we can observe that, the impacts of memory limitation on the mean of time to failure of Airline and Account are greater than that of concurrency level. Comparatively, we can see from Fig. 2c that, the increasing of concurrency levels can decrease the time to datarace failure rather rapidly than memory limitation. Thus, combining with the Kruskal-Wallis tests results, we can conclude that, both memory limitation and concurrency level have impacts on the MTTF of the subjects. The impacts of different factors on the same program suffering a datarace bug are different.

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

This research question is to figure out the fittest relationship model between the MTTF and the influencing factors once the MTTF is significantly affected. In this work, we test four types of models:

- **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 MTTF variable, \( S \) denotes influencing factor variable in our work, \( \beta_0 \) denotes the intercept parameter, \( \beta_1 \) denotes the coefficient parameter, and \( \log(\cdot) \) is the natural logarithm function in our whole context.

We use the least squares method to fit these models. We adopt R-squared and R-squared adjust as the goodness of fit (GoF) measure [21]. The larger the value of R-squared or R-squared adjust is, the better fit of the model is.

Table III summarizes the least squares fitting results. For Airline, its fittest model to the impact of memory limitation is Power model. For Account, both Power model and Exponential model can fit well to the impact of memory limitation. For MySQL, its fittest model to the impact of concurrency level is Power model. In Table III, the last column presents the fittest MTTF regression models with respect to influencing factor variable, where \( ML \) denotes maximum limitation variable, \( CL \) denotes concurrency level variable. Thus, in this part, we observe that Power model is a good one to fit the impacts for all subject.

Fig. 4a illustrates line of the logarithmically transformed mean time to failure (MTTF (transformed)) of Airline with respect to logarithmically transformed memory limitation variable (ML (transformed)). It can be observed that there is a significant upward trend of MTTF (transformed) with respect to the increasing of memory limitation for Airline. Fig. 5a illustrates the line of MTTF (transformed) of Account with respect to memory limitation variable. It can be observed that there is a significant upward trend of MTTF (transformed) with the increasing of memory limitation for Account. Comparing Fig. 4a with Fig. 5a, we can observe that, although the slopes for the trends, memory limitation can work as a controllable factor for the reducing of MTTF.

Fig. 6a illustrates the line of the MTTF (transformed) of MySQL with respect to logarithmically transformed concurrency level (CL (transformed)). It can be observed that there is a significant downward trend of MTTF (transformed) with respect to the increasing of concurrency levels. That means, concurrency level can also work as a controllable factor for the reducing of mean time to a datarace failure. Recall that, for Airline and Account, because there is not enough statistical data collected, the observation is only based on the experiments on MySQL.

**RQ3: Is the distribution model of TTF changing with the variation of influencing factors?**

We select Weibull, LogNormal, Exponential, and Normal distributions [22] as the candidate distribution models. These distributions are commonly used in reliability area for modeling the life of programs. We adopted the Anderson-Darling adjusted (AD) [23] statistic as goodness of fit (GoF) measure of these models by analyzing the raw experimental data. For AD, the value closer to zero indicates better fit. According to the result of RQ1, we made fitting on the affected levels, i.e., totally 18 levels.

Table IV summarizes the distribution fitting results. The second to the fifth columns shows the number of each type of distribution as the fittest model. From this table, we can
where $\beta$ denotes the shape parameter, $\eta$ denotes the scale parameter.

Fig. 4b shows the Airline TTF’s 90, 50, and 10 percentile of lines with respect to the memory limitations. From this figure, we can see the experimental outputs fall into the fitted lines. Fig. 4c shows the standardized residuals plots between the Power and Exponential models. Since the distribution model we used for the maximum likelihood estimation is Weibull distribution, it is expected that the standardized residuals can be similar to the samples from standardized Smallest-Extreme distribution. From Fig. 4c, we can observe that the standardized residuals almost fall into the 95% confidence intervals, indicating the adequacy of estimated model. Table V shows the estimation results for Airline, where $\sigma$ is the scale parameter. The pdf of Airline’s TTF is given by Power-Weibull distribution, i.e., equation (3), by letting $\beta$ equal $1/\sigma$ and $\eta$ equal $T$ [22], in which $T$ is Power model, for equation (2). The values and intervals of $\beta_0, \beta_1,$ and $\sigma$ are shown in Table V.

$$f(t; ML) = \frac{1}{\sigma^{\beta_1}ML^{\beta_0}} \left( \frac{t}{\beta_0ML^{\beta_1}} \right)^{\frac{1}{\beta_0} - 1} e^{-\left( \frac{t}{\beta_0ML^{\beta_1}} \right)^{\frac{1}{\beta_0}}} \right)^{\frac{1}{\beta_0}} \right)$$

(3)

where $ML$ denotes memory limitation variable, $t$ denotes TTF variable of Airline.

For Account, according to Table IV, its fittest distribution is LogNormal whose pdf is given by equation (2). LogNormal distribution is skewed distribution and its logarithmic transformation is Normal distribution, and we need only to compare standardized residuals plots between the Power and Exponential models [22], and we found that Exponential model is the better one, which is consistent with the result obtained by the least squares method in RQ2.

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

(4)

where $\mu$ denotes the location parameter, $\sigma$ denotes the scale parameter, and $\log(\cdot)$ is the natural logarithm function.

Fig. 5b shows the Account TTF’s 90, 50, and 10 percentile of lines with respect to the memory limitation. From this figure, we can see the experimental outputs fall into the fitted lines. Fig. 5c shows the standardized residuals plots. Since the distribution model we adopted for maximum likelihood estimation is LogNormal distribution, it is expected that the standardized
(a) MTTF (transformed) of Airline with respect to memory limitation variable (transformed)

(b) Airline TTF’s percentile plot with respect to the memory limitation variable

(c) Standardized residuals of Smallest-Extreme distribution and its 95% CI plot

(a) MTTF (transformed) of Account with respect to memory limitation variable

(b) Account TTF’s percentile plot with respect to the memory limitation variable

(c) Standardized residuals of Normal distribution and its 95% CI plot

Fig. 4: Airline Plots

Fig. 5: Account Plots
residuals can be similar to the samples from standardized Normal distribution. From Fig. 5c, we can observe that the standardized residuals almost fall into the 95% confidence intervals, indicating the adequacy of estimated model. Table VI shows the estimation results for Account, where $\sigma$ is the scale parameter. The pdf of Account’s TTF is given by Exponential-LogNormal distribution, i.e., equation (5), by letting $\mu$ equal $\log(T)$ [22], in which $T$ is Exponential model, for equation (4). The values and intervals of $\beta_0$, $\beta_1$, and $\sigma$ are shown in Table VI.

$$f(t; ML) = \frac{1}{t \sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{\log(t) - \log(\beta_0) - \beta_1 ML}{\sigma} \right)^2}$$

(5)

where $ML$ denotes memory limitation variable, $t$ denotes the TTF variable of Account.

---

**TABLE VI: Maximum Likelihood Estimation Results of Account**

<table>
<thead>
<tr>
<th>Param.</th>
<th>Estimator</th>
<th>Std. Error</th>
<th>95% Normal C.I. Lower-B</th>
<th>95% Normal C.I. Upper-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log(\beta_0)$</td>
<td>1.51013</td>
<td>0.416449</td>
<td>0.693902</td>
<td>2.32635</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>0.0615714</td>
<td>0.0073238</td>
<td>0.0472170</td>
<td>0.0759258</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>1.37016</td>
<td>0.10216</td>
<td>1.18393</td>
<td>1.58568</td>
</tr>
</tbody>
</table>

---

For MySQL, according to Table IV, its fittest distribution is LogNormal. By comparing standardized residuals plots between the Power and Exponential models, we found that Power model is the better one, which is consistent with the result obtained by the least squares method in RQ2. Fig. 6b shows the MySQL TTF’s 90, 50, and 10 percentile of lines with respect to concurrency level variable. From this figure, we can see the experimental data fall in the fitted lines. Fig. 6c shows the standardized residuals of the experimental outputs. Since the distribution model we adopted for maximum likelihood estimation is LogNormal distribution, it is expected that the standardized residuals can be similar to the samples from standardized Normal distribution. From Fig. 6c, we can observe that the standardized residuals almost fall into the 95% confidence intervals, indicating the adequacy of estimated model. Table VII shows the estimation results for MySQL, where $\sigma$ is the scale parameter. The pdf of MySQL’s TTF is given by Power-LogNormal distribution, i.e., equation (6), by letting $\mu = \log(T)$ [22], in which $T$ is Power model, for equation (4). The values and intervals of $\beta_0$, $\beta_1$, and $\sigma$ are shown in Table VII.

$$f(t; CL) = \frac{1}{t \sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{\log(t) - \log(\beta_0) - \beta_1 \log(CL)}{\sigma} \right)^2}$$

(6)

where $CL$ denotes the concurrency level variable, $t$ denotes the TTF variable of MySQL.

In this part, we analyzed the TTF model with respect to influencing factors for each subject. From the analysis results, we can observe that, the fittest relationship models used for maximum likelihood estimation are consistent with the least squares regression analysis results in RQ2, which are summarized in Table III.
TABLE VII: Maximum Likelihood Estimation Results of MySQL

<table>
<thead>
<tr>
<th>Param.</th>
<th>Estimator</th>
<th>Std. Err.</th>
<th>95% Normal C.I. Lower-B</th>
<th>95% Normal C.I. Upper-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log(\beta_0)$</td>
<td>12.03194</td>
<td>0.317132</td>
<td>11.3978</td>
<td>12.6140</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>-1.99602</td>
<td>0.0885512</td>
<td>-2.19500</td>
<td>-1.79704</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>1.38133</td>
<td>0.102391</td>
<td>1.19454</td>
<td>1.59732</td>
</tr>
</tbody>
</table>

RQ5: How to estimate the MTTF?

Based on the results of RQ3 and RQ4, we further explore the method to estimate the MTTF.

For Airline, the fittest distribution is Weibull distribution, whose MTTF is given by equation (7) [24]. In equation (7), the shape parameter $\beta$ should be $1/\sigma$, and the scale parameter $\eta$ should be substituted with the Power model, i.e., $\beta_0 ML^{\beta_1}$. The values and intervals of $\sigma$, $\beta_0$, and $\beta_1$ are given in Table V. After simple calculation, the MTTF of Airline is given by (I) $MTTF_{Airline} = 0.9146 * e^{0.864067 * ML^{0.8073}}$, where $ML$ denotes the memory limitation variable.

$$MTTF = \eta \cdot \Gamma \left(1 + \frac{1}{\beta}\right)$$ (7)

where $\beta$ denotes the shape parameter, $\eta$ denotes the scale parameter, and $\Gamma(\cdot)$ denotes the Gamma function.

For Account, the fittest distribution is LogNormal distribution, whose MTTF is given by equation (8) [24]. In equation (8), the location parameter $\mu$ should be substituted with the Exponential model, i.e., $\beta_0 e^{\beta_1 ML}$. The values and intervals of $\sigma$, $\beta_0$, and $\beta_1$ are given in Table VI. After simple calculation, the MTTF of Account is given by (II) $MTTF_{Account} = e^{2.4488+0.0616*ML}$, where $ML$ denotes the memory limitation variable.

$$MTTF = \exp(\mu + \frac{1}{2}\sigma^2)$$ (8)

where $\mu$ denotes the location parameter, and $\sigma$ denotes the scale parameter.

For MySQL, the fittest distribution is LogNormal distribution, whose MTTF is given by equation (8). In equation (8), the location parameter $\mu$ should be substituted with the Power model, i.e., $\beta_0 CL^{\beta_1}$. The values and intervals of $\sigma$, $\beta_0$, and $\beta_1$ are given in Table VII. After simple calculation, the MTTF of MySQL is given by (III) $MTTF_{MySQL} = e^{12.9734 * CL^{-1.90602}}$, where $CL$ denotes the concurrency level variable.

By comparing between the three MTTF functions estimated in the above part, i.e., (I), (II), and (III), and the MTTF regression results in Table III, we found that the major differences are their intercept parameters. These differences are caused by whether using the information of the TTF distribution model or not. Since the distribution model information is used in this part in the maximum likelihood estimation method, the MTTF estimation functions should be more accurate comparing to the results obtained in RQ2. As a result, we can conclude that the approach can be used to estimate the MTTF.

VI. THREATS TO VALIDITY

In the experiments, we used two benchmark programs and MySQL to verify the factors. These experimental programs belong to desktop applications (Airline, Account, and MySQL) that are programmed by Java or C++ language. Further experiment on widely used applications such as Firefox and Apache Web Server may strengthen the conclusions. Besides, the time measurement may be affected if other programming languages were used for implementation.

Our hypotheses test method used in this work is Kruskal-Wallis test. The hypotheses test results may be affected if other techniques are used. The fitting techniques used in this work are least squares regression and maximum likelihood estimation implemented by Minitab. The fitting results could be affected if other techniques are used.

Except for memory limitation and concurrency level, there could be other factors which can be used to accelerate the occurrence of data race failures. Systematically studying all the influencing factors for data race failures will be our work in the near future.

VI. RELATED WORK

There have been several studies on the acceleration of software failures. Chan [13] proposed the concept of using accelerated stress testing on both hardware and software. Gullo et al. [25] proposed to use accelerated stress testing to reduce detecting time for probabilistic software bugs. Gullo et al. applied “increasing the operational frequency (use-rate)” as the acceleration method to accelerate the failure process of their system. They did not show the acceleration model between accelerating variable and the failure time.

The acceleration is especially required for the programs suffering bugs whose manifestation is dependent on the execution environment, for example aging related bugs [26], the time to a failure triggered by those bugs is usually very long. Considering the difficulty, several researches have started to study the acceleration approaches and correspondingly MTTF estimation techniques. For example, Cavezza et al. [27] explored three possible environmental accelerating factors for MySQL software. Matias et al. [28] developed a systematic approach to accelerate the aging effects at the experimental level. They introduced the concept of aging factors and used different levels of accelerated workload to increase the system degradation rate. Based on the degradation data of selected system characteristics, captured through measurements, they apply the statistical technique of accelerated degradation tests (ADT) to estimate the time to failure in normal condition (without acceleration). In Matias et al. [10], the authors used accelerated life tests (ALT) to estimate the time to failure in normal conditions by observing failures obtained under accelerated workloads. In [29], in order to reduce the time to application failures caused by memory leaks, Zhao et al. select the Weibull time to failure distribution at normal level and use the accelerated life testing (ALT) approach to estimate the MTTF. They further use a semi-Markov process to compute the optimal software rejuvenation trigger interval. The existent work shows the effectiveness of applying AT approaches (ADT or ALT) into the acceleration of failures triggered by aging related bugs and also the estimation of MTTF due to the aging.
related bug triggered failures. Aging related bug is a fault that leads to the accumulation of errors either inside the running application or in its system-context environment, resulting in an increased failure rate and/or degraded performance.

Comparing with aging-related bugs, for a datarace bug, usually there is not any sign before the failure. The bugs activation is non-deterministic and once the fault is triggered, the failure will manifest rapidly. Therefore, the accelerating factor selection for accelerating the aging failure process cannot be used for datarace. Consequently, the acceleration models which relate the failure process with respect to the accelerating variable cannot be applied either. To the best of our knowledge, this is the first work to study the influence of influencing factors on time to a datarace failure, and this is also the first work to explore the way of estimating the MTTF of datarace. Due to the significance and popularity of datarace bugs, we believe the topic is deserved to be studied in depth.

VII. CONCLUSIONS AND FUTURE WORK

In this work, we selected the influencing factors, i.e., memory limitation and concurrency level, to accelerate the time to a datarace failure for a specific program and further studied the influences on the TTF and MTTF. We conducted experiments on real datarace suffering programs to verify the factors and their influences. Through the ANOVA analysis of experimental results, i.e., RQ1, we observed that both memory limitation and concurrency level have impacts on the MTTF of the subjects, and the impacts of different factors on the same program are different. Then we further studied the relationship model with MTTF, i.e., RQ2, we observed that both memory limitation and concurrency level can work as controllable factors for the reducing of MTTF, and Power model is a good one to fit their impacts among different models. We studied the distribution model of TTF, i.e., RQ3, and observed that fittest distribution model changes with the variation of the level of influencing factors. We studied the estimation of TTF distribution with respect to influencing factors, i.e., RQ4. Based on the answers of RQ3 and RQ4, we gave the MTTF estimation with respect to influencing factors for each experimental subject, i.e., RQ5. By comparing RQ2 answers with RQ5 answers, we found the MTTF estimation results are consistent with the regression results, we finally conclude that two factors can be used to accelerate a datarace failure and accurately estimate the MTTF.

Our work mainly have three contributions: (1) It is the first work to study the impacts of influencing factors on the time to a datarace failure. (2) The influences on TTF and MTTF of a datarace software failure are characterized from the statistical perspectives, which are important for software reliability or availability modeling studies [7], [8], [9]. (3) We apply memory limitation and concurrency level as the influencing factors and empirically verified their relationship models with the MTTF, which should be helpful for the accelerating life testing work in software reliability engineering area [10], [29].

In the near future, more influencing factors, multiple datarace bugs in a program and different types of workloads will be considered in our empirical study.

ACKNOWLEDGEMENT

We would like to thank Jing Zhao and Nan Wang for their helpful discussions. This work is supported by the National Natural Science Foundation of China (Grant NO. 61572150, 61402027, and 61772055), and in part by US NSF (Grant NO. CNS-1523994).

REFERENCES


