A Study of Concurrency Bugs in an Open Source Software

Sara Abbaspour Asadollah[1], Daniel Sundmark[1], Sigrid Eldh[2], Hans Hansson[1], and
Eduard Paul Enoiu[1]

[1] Mälardalen University, Västerås, Sweden
{sara.abbaspour, daniel.sundmark, hans.hansson,
eduard.paul.enoiu}@mdh.se
[2] Ericsson AB, Kista, Sweden {sigrid.eldh@ericsson.com}

Abstract. Concurrent programming puts demands on software debugging and testing, as concurrent software may exhibit problems not present in sequential software, e.g., deadlocks and race conditions. In aiming to increase efficiency and effectiveness of debugging and bug-fixing for concurrent software, a deep understanding of concurrency bugs, their frequency and fixing-times would be helpful. Similarly, to design effective tools and techniques for testing and debugging concurrent software understanding the differences between non-concurrency and concurrency bugs in real-word software would be useful. This paper presents an empirical study focusing on understanding the differences and similarities between concurrency bugs and other bugs, as well as the differences among various concurrency bug types in terms of their severity and their fixing time. Our basis is a comprehensive analysis of bug reports covering several generations of an open source software system. The analysis involves a total of 4872 bug reports from the last decade, including 221 reports related to concurrency bugs. We found that concurrency bugs are different from other bugs in terms of their fixing time and their severity. Our findings shed light on concurrency bugs and could thereby influence future design and development of concurrent software, their debugging and testing, as well as related tools.

Keywords: Concurrency bugs·Bug severity·Fixing time·Open source software.

1 Introduction

With the introduction of multicore and other parallel architectures, there is an increased need for efficient and effective handling of software executing on such architectures. An important aspect in this context is to understand the bugs that occur due to parallel and concurrent execution of software. In this paper we look into how the increase of such executions have impacted a number of issues, including the occurrence of related bugs and the difficulty to fix these bugs compared to fixing non-concurrent ones.

Testing and debugging concurrent software are faced with a variety of challenges [1]. These challenges concern different aspects of software testing and debugging, such as parallel programming [2], performance testing, error detection [3] and more. Since concurrent software exhibit more non-deterministic behavior and non-deterministic bugs are generally viewed to be more challenging than other types of bugs [4, 5, 6], testing and debugging concurrent software are also considered to be more challenging compared to testing and debugging of sequential software.

Developing concurrent software requires developers to keep track of all the possible communication patterns that evolve from the large number of possible interleavings or concurrently overlapping executions that can occur between different execution threads through utilizing the shared memory.

Handling the many execution scenarios that this results in is a notoriously difficult task in debugging and makes it equally hard to create test cases [7].

In the study presented in this paper we are particularly interested in isolating concurrency bugs from other types of bugs (non-concurrency bugs) and analyzing the distinguishing features in their respective fixing processes. Hence, the main emphasis of this research is on concurrency bugs, and to explore the nature and extent of concurrency bugs in real-world software. This exploration of bugs can be helpful to understand how we should address concurrency bugs, estimate the most time-consuming ones, and prioritize them to speed up the debugging and bug-fixing processes. Also it could be helpful for designers to avoid the errors that are more likely to occur during the early phases of the software lifecycle.

In our study we address the following research questions:

– **RQ1:** How common are different types of concurrency bugs, compared to non-concurrency bugs?
– **RQ2:** How much time is required to fix concurrency bugs, compared to fixing non-concurrency bugs?
– **RQ3:** Are concurrency bugs more severe than non-concurrency bugs?

In this study we investigate the bug reports from an open source software project. We classify bugs into two distinct groups, i.e., concurrency bugs and non-concurrency bugs. We classify the concurrency bugs based on bug type, severity and fixing time. We compare the non-concurrency and concurrency bug in terms of their reporting frequency, severity and fixing time. Our results indicate that a relatively small share of bugs is related to concurrency issues, while the vast majority are non-concurrency bugs. Fixing time for concurrency and non-concurrency bugs is different but this difference is not big. In addition, concurrency bugs are considered to be slightly more severe than non-concurrency bugs.

2 Methodology

In this study first we start with *Bug-source software selection* in order to select a proper open source software for our study. Second, we identify the set of concurrency bug reports in the issue tracking database of the selected project through a keyword search in *Bug report selection*. Then we manually analyze the full set of identified bug reports in order to exclude those that are not concurrency-related. Finally, in *bug reports classification* process, we collect data for the concurrency bugs, and classify the bug reports based using the classification scheme described in Section 3. The following subsections describe the steps of this research process in further detail.

2.1 Bug-source Software Selection

We were interested in an open source application that coordinates distributed processes with significant number of releases and an issue management platform for managing,

configuring and testing. We selected the Apache Hadoop project[1] as the open source project for our study. The full justification for selecting Hadoop as our study object is provided in the list below.

(1) Hadoop has changed constantly and considerably in 59 releases over six years of development. (2) Due to Hadoop's key concept of parallel and distributed abstraction it is recently adopted by several big companies (i.e., Facebook, Ebay, Yahoo, Amazon and more). (3) Detailed information on bugs and bug fixes are openly available. (4) The Hadoop framework has been widely adopted by both the industry and research communities [8]. (5) It has a web interface for managing, configuring and testing its services and components.

Hadoop tracks both enhancement requests and bugs using JIRA[2]. JIRA is an issue management platform, which allows users to manage their issues throughout their entire lifecycle. It is mainly used in software development and allows users to track any kind of unit of work, such as project task, issue, story and bug to manage and track development efforts.

2.2 Bug Reports Selection

In this stage we selected the concurrency bugs from the bug report database including bugs from different versions of Apache Hadoop, including bug reports from the period 2006-2015, i.e., the last decade. In total, the Hadoop bug report database contains 4872 issues in this period that are tagged as "Bug"[3].

We automatically filtered reports that are not likely to be relevant by performing a search query on the bug report database. Our search query filtered bugs based on (1) "Bug" as report type, (2) the status of the report, and (3) keywords relevant to concurrency. Figure 1 summarizes the bug report selection process.

In filtering based on *"Bug" as report type* step, we practically searched in the Apache Hadoop report database for the reports with issue type "Bug" according to our main objective and bug definition.

In filtering based on *the status of the report* step, we searched for bugs with "Closed" (i.e., this report considered finished, the resolution is correct) and "Fixed" resolution status (i.e., fix for this issue has been implemented). We only selected "Fixed" and "Closed" reports since unfixed and open bug reports might be invalid and root causes described in the reports could be incorrect. It would then be impossible for us to completely understand the details on these bugs and determine their types.

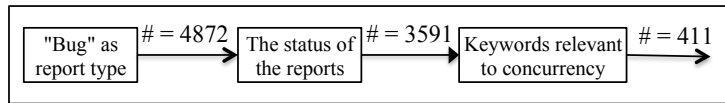| "Bug" as report type | # = 4872 | The status of the reports | # = 3591 | Keywords relevant to concurrency | # = 411 |

Fig. 1: Bug report selection workflow

In filtering based on *the keywords relevant to concurrency* step, we decided to use the keywords that could help us to include the bug reports were compatible with the scope of this study. In identifying such keywords, we reviewed the keywords utilized

---

[1] https://issues.apache.org/jira/browse/hadoop

[2] https://www.atlassian.com/software/jira

[3] Bug is "a problem which impairs or prevents the functions of the product" [9].

in similar previous studies [10, 1]. The keywords included in the search, i.e. the terms, are as follows. After filtering we obtained a final set with 411 reports.

thread, blocked, locked, race, dead-lock, deadlock, concurrent, concurrency, atomic, synchronize, synchronous, synchronization, starvation, suspension, "order violation", "atomicity violation", "single variable atomicity violation", "multi variable atomicity violation", livelock, live-lock, multi-threaded, multithreading, and multi-thread.

Table 1 shows the bug count across the different stages of the bug report selection process. Note that this selection process may have some limitations, discussed in more detail in subsection 5.1.

Table 1: Report counts from different stage of bug report selection process

Filter	Selected reports	# of reports
	Total Hadoop bug reports	3591
2006-2015 & Bug & Fixed & Closed	Keywords match related bug reports	411
	Concurrency bug reports analyzed	221
2006-2015 & Bug & Fixed & Closed	Sample of **non-concurrency** bug reports	221

2.3 Manual Exclusion of Bug Reports and Sampling of Non-concurrency Bugs

In this stage we manually analyzed the 411 bug reports obtained in the previous step[4]. The manual inspection revealed that some of the bugs that matched the search query were not concurrency bugs. Thus, we excluded them. More specifically, we determined the relevance of the bugs by checking (1) if they describe a concurrency bug, and if they do, (2) what type of concurrency bug is it. The latter is done, by comparing their descriptions (or explanations) with our concurrency bug classification (Section 3.1). If we could not map a report with any class we excluded that report from our set. We also excluded reports with very little information, since we could not analyze them properly. After filtering we obtained a final set with 221 concurrency bugs.

As explained in Section 1, our main objective is understanding the differences between non-concurrency and concurrency bugs. For comparison purposes, we randomly sampled an equally sized subset of non-concurrency bugs that were reported during 2006-2015 and were "Fixed" and "Closed". These bugs were used for comparative analysis between the concurrency and non-concurrency bug sets. In this study, we use the term *non-concurrency bugs* instead of *sample of non-concurrency bugs* for all comparative analysis.

2.4 Bug Reports Classification

We analyzed the issues and information contained in the reports using them to map to the concurrency bug classification manually. Each bug report contains several types of information, which were valuable in recognizing and filtering the concurrency bugs with other types of bugs to aids us understand the characteristics of bugs. The bug reports contained for example the description of the bug with some discussions among the developers on how to detect, where to detect (bug localization) and how to fix the bugs. Typically most of the reports include a description of the correction of the bug, and a link to the the version of the software where the bug has been corrected, and even the scenario of reproducing the reported bug. The reports also contain additional fields such as perceived priority, created date, resolved date, version affected and more.

---

[4] We provide the raw data of this study at https://goo.gl/sr6iDQ.

We used different types of fields in order to explore the concurrency bug issues in the Hadoop project. We used the *priority* field to estimate the severity of the bug. The interval between the *Created date* and *Resolved date* fields was used to calculate the amount of (calendar) time required to fix the bug (fixing time).

3 Study Classification Schemes

In order to perform the bug classification process we defined three main classifiers and grouped the reports based on these classifiers. The classifiers were **Type of concurrency bug**, **fixing time** and **severity**. These three classification schemes are described in detail below.

3.1 Concurrency Bug Classification

In [10], our main contribution is a better understanding of the different types of concurrency bugs. We classified and mapped the relevant bug reports related to the types of concurrency bugs using a classification of concurrency bug types. It categorizes concurrency bugs into seven disjoint classes (i.e., ***Deadlock***, ***Livelock***, ***Starvation***, ***Suspension***, ***Data race***, ***Order violation*** and ***Atomicity violation***).

3.2 Fixing Time Calculation

This class shows the time duration (days) which developer (or debugger) spend to fix a reported bug. ***Fixing time***, calculated by subtracting the *Created date* and *Resolved date* fields of the bug.

3.3 Bug Report Severity Classification

In order to define priority for each issue based on developers' perspective we used a classification scheme similar to the classification defined in [9]. ***Blocker*** shows the highest priority. It indicates that this issue takes precedence over all others. ***Critical*** indicates that this issue is causing a problem and requires urgent attention. ***Major*** shows that this issue has a significant impact. ***Minor*** indicates that this issue has a relatively minor impact. ***Trivial*** is the lowest priority.

4 Results and Quantitative Analysis

This section provides the analysis of the data collected for bugs obtained from the Hadoop project bug database. We used 442 bugs (i.e., 221 are concurrency bugs while the rest are non-concurrency bugs sampled for our analysis) reported between 2006 and 2015. The bug selection process is described in Section 2.

RQ1: How common are different types of concurrency bugs, compared to non-concurrency bugs?

As seen in Figure 2(a), out of the 3591 bugs reported in the Hadoop database, 221 (i.e., 6.15%) bugs are related to concurrency issues and are causing a certain type of concurrency bugs, while the rest (i.e., 93.85%) are identified as non-concurrency bugs.

The 221 concurrency bugs were further categorized according to the concurrency bug classification in [10]. As mentioned already in Section 3.1, this taxonomy defines seven types of concurrency bugs. For the sake of this study, we have added *Not clear* category to the taxonomy. The *Not clear* category includes reports that cover bugs related to concurrency and parallelism, but are not classified according to the concurrency bugs taxonomy. For these bugs, the summary and description of the report shows it is a concurrency bug, but further classification of bug type is prohibited by a very project implementation-specific explanation of the bug details and solution.

In addition, we investigated the frequency with which each type of concurrency bug appears, with the aim of getting insights into bug prioritization. In Figure 2(b) we show the number of concurrency bugs according to their category and how often they are reported in the data we collected. From a total of 221 bug reports, almost half of them (i.e., 44%) concern data races (or race conditions), a well-known and common concurrent bug [11]. In addition, about 15% of the reports reported the *Suspension* bug type and only two bug reports were categorized as *Livelock* bugs.
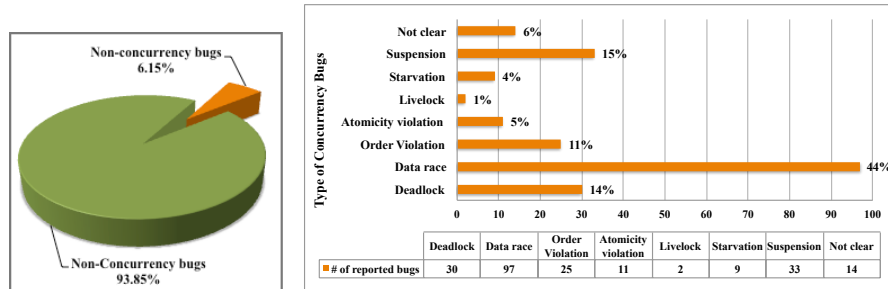
(a) Distribution of non-concurrency and concurrency bug types

(b) Distribution of concurrency bugs

Fig. 2: Distribution of bugs

Answer RQ1: Only 6.15% of the total set of bugs are related to concurrency issues, while the majority of bugs (i.e., 93.85%) are of non-concurrency type.

RQ2: How much time is required to fix concurrency bugs, compared to fixing non-concurrency bugs?

In order to gain better understanding on how difficult is to fix concurrency bugs in comparison with non-concurrency bugs, we conducted a quantitative analysis of the effort required to fix both concurrency and non-concurrency bugs. This effort was measured as explained in Section 3.2. We used this time as an indicator for the complexity involved in fixing these bugs.

Table 2(a) lists the detailed statistics on the obtained results for fixing time of concurrency and non-concurrency bugs. These results are also summarized in Figure 3(a) in the form of box-plots (the vertical axis scale of the plot is logarithmic). Interestingly, the fixing time for concurrency and non-concurrency bugs is very similar, with an average of 58 days and 54 days for fixing concurrency and non-concurrency bugs, respectively.

To evaluate if there is any statistical difference between concurrency and non-concurrency bugs fixing time we use a *Wilcoxon Signed Rank* test, a non-parametric hypothesis test for determining if there is any statistical difference among two data sets, with the assumption that the data is drawn from an unknown distribution. We use 0.05 as the significance level.

Table 2: Descriptive statistics results for concurrency and non-concurrency bug sets in terms of fixing time.

(a) Fixing time comparison

| Fixing time / Bug	Average	Minimum	Maximum	Median	Standard Deviation
Concurrency	58.3	0.1	1221.0	13.1	143.4
Non-concurrency	54.2	0.1	998.1	7.9	133.3

(b) Fixing time comparison for concurrency bugs

Fixing time / Concurrency bug	Average	Minimum	Maximum	Median	Standard Deviation
Deadlock	43.1	0.1	943.2	4.3	171.5
Data race	80.0	0.1	1221.0	17.0	181.4
Order violation	54.9	0.1	471.3	20.4	100.6
Atomicity violation	39.1	0.2	253.7	19.7	72.3
Livelock	16.9	15.0	18.9	16.9	2.7
Starvation	24.4	1.0	89.2	12.8	29.6
Suspension	38.5	0.5	197.2	17.9	46.9

In addition, we calculate the *Vargha-Delaney A-statistic* as a measure of effect size [12] for analyzing significance. This statistic is independent of the sample size and has a range between 0 and 1. The choice of what constitutes a significant effect size can depend on context. Vargha and Delaney [12] suggest that A-statistic of greater than 0.64 (or less than 0.36) is indicative of "medium" effect size, and of greater than 0.71 (or less than 0.29) can be indicative of a "large" effect size.

We are interested in determining if the fixing time for concurrency bugs is similar to the one for non-concurrency bugs. We begin by formulating the statistical hypotheses as follows: the null hypothesis is that fixing time of the concurrency and non-concurrency bugs sets have identical distributions (H_0) and the alternative hypothesis is that the distributions are different (H_1). Based on the p-value of *0.027*, which is less than 0.05, we reject the null hypothesis. That is, the fixing time of concurrency bugs and non-concurrency bugs are statistically different. When calculating the Vargha-Delaney A-statistic we obtained a value of *0.560* which indicates a "small" standardized effect size [12]. From our results, we can see that the fixing time for concurrency bugs is different from the fixing time for non-concurrency bugs, but that this difference corresponds to a "small" standardized effect size.

We were also interested in understanding the differences between fixing time for each type of concurrency bugs. Figure 3(b) summarize our results in the form of box plots. It is obvious that *Data races* took the longest time to fix (i.e., 80 days on average). *Order violation* and *Deadlock* type of bugs took less time (55 and 43 on average, respectively) while *Livelock* and *Starvation* type of bugs took shorter fixing time (17 and 24 days on average, respectively). Table 2(b) lists the detailed statistics on the obtained results for each type of concurrency bugs. To evaluate if there is any signif-
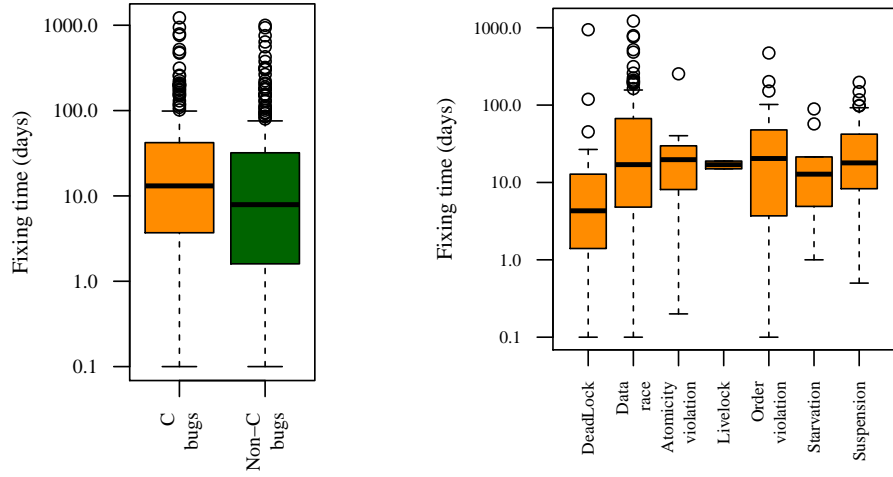
(a) Fixing time comparison for concurrency (C bug) and non-concurrency bugs (Non-C bugs).

(b) Effort required to fix each type of concurrency bugs.

Fig. 3: Fixing time analysis; boxes span from 1st to 3rd quartile, black middle lines are marking the median and the whiskers extend up to 1.5x the inter-quartile range while the circles represent the outliers.

icant statistical difference between the different types of concurrency bugs, we use a *Wilcoxon Signed Rank test* and calculate the A-statistic effect size. To this end, we report in Table 3 the p-values and the effect size for each type of concurrency bugs. The tested hypotheses are formulated as follows: the null hypothesis is that fixing time results between two different bug types are drawn from the same distribution and the alternative hypothesis is that the fixing time results are drawn from different distributions. We use a traditional statistical significance limit of 0.05 and Vargha and Delaney's suggestion [12] for statistical significance. Examining Table 3, we can conclude that the null hypothesis is accepted with p-values above the traditional statistical significance limit of 0.05 for the majority of bug types except for *"Deadlock-Data race"* and *"Deadlock-Suspension"* pairs where the null hypothesis is rejected. This shows that the bug fixing time is not different except between "Deadlock-Data race" and "deadlock-Suspension". For example, in Table 3 we show the obtained p-value of *0.008* for testing the pair "Deadlock-Data race", which is less than 0.05, and therefore we can reject the null hypothesis: the fixing time for *Deadlock* and *Data Race* bug types are different. In addition, the A-statistic for the same pair of bug types is about *0.796* (or 0.204 in the second row), which is greater than the significance level of 0.71. We can say that in this case the effect size is "large". We can conclude that fixing time for *deadlock* and *data race* bug types is different with a "large" effect size.

It should however be noted that the likelihood of statistical errors vastly increases when doing multiple tests using the same dataset. The results from the inter-bug-type comparisons are thus less reliable than the results from the comparison between concurrency and non-concurrency bugs.

Table 3: Wilcoxon test for concurrency bugs fixing time comparison

H₀ \ Hₐ	Hypothesis test result	Deadlock	Data race	Order violation	Atomicity violation	Livelock	Starvation	Suspension
Deadlock	P-value	-	0.007389	0.03635	0.06946	0.1476	0.128	0.001921
	A-statistic	-	0.7951961	0.2046976	0.0892762	0.0154108	0.0735466	0.2636306
Data race	P-value	0.007389	-	0.9368	0.9595	0.9702	0.646	0.5778
	A-statistic	0.2048039	-	0.1302476	0.0572856	0.0100967	0.0507493	0.1589967
Order violation	P-value	0.03635	0.9368	-	0.9452	0.8894	0.6391	0.6888
	A-statistic	0.2826018	0.8697524	-	0.0990541	0.0181741	0.0821554	0.2936019
Atomicity violation	P-value	0.06946	0.9595	0.9452	-	0.9231	0.7902	0.6644
	A-statistic	0.2983314	0.9427144	0.2428526	-	0.0195557	0.0879477	0.3191625
Livelock	P-value	0.1476	0.9702	0.8894	0.9231	-	0.8128	0.8869
	A-statistic	0.3081093	0.9899033	0.2548624	0.1121267	-	0.0919864	0.3364332
Starvation	P-value	0.128	0.646	0.6391	0.7902	0.8128	-	0.4343
	A-statistic	0.2998193	0.9492507	0.2444468	0.107716	0.0195026	-	0.3216601
Suspension	P-value	0.001921	0.5778	0.6888	0.6644	0.8869	0.4343	-
	A-statistic	0.2806356	0.8410033	0.2166543	0.0958657	0.017377	0.0797641	-

> *Answer RQ2: Concurrency bugs do require longer fixing time than non-concurrency bugs, but the difference is not very large.*

RQ3: Are concurrency bugs more severe than non-concurrency bugs?

We analyzed the difference between concurrency and non-concurrency bug severity in order to understand if the severity of bugs is differently distributed. Figure 4 shows the severity distributions. In order to statistically compare the severity between concurrency bugs and non-concurrency bugs, we apply a *Two-Sample Kolmogorov-Smirnov* test (also known as two-sample K-S test) to find if the frequency between these two types of bugs is significantly different. Our null hypothesis can be formulated as follows: are the severity level results of concurrency bugs and non-concurrency bugs drawn from the same distribution. In this test, if the *D-value* is larger than the *critical-D-value*, the observed frequency is distributed differently.
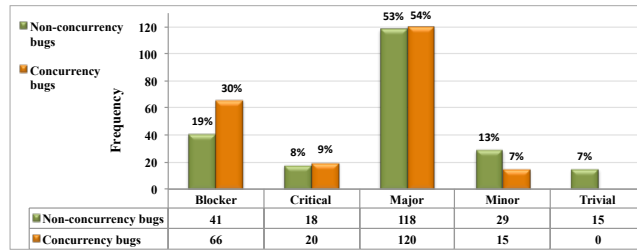

Fig. 4: Concurrency and non-concurrency bug severity

Table 4 shows that the D-value is *0.131*, which is larger than the Critical-D-value of *0.091*. Thus, statistically we have enough evidence to conclude that there is a difference between the concurrency and non-concurrency bug severity distribution. In other words, the concurrency and non-concurrency severity types are distributed differently.

Finally, we are also interested to identify the severity distribution difference between different concurrency bugs classes. The results obtained for this analysis are shown in Figure 5. The results indicate that the highest severity is observed for the "Blocker" class. We expected that most of the bugs to be of *Deadlock* type. In reality, as shown in Figure 5, most of the bugs are of *Data race* type. We can interpret this fact in the following way: the *Data race* type might represent the most problematic bug type in terms of severity in the Hadoop project.

Table 4: Kolmogorov-Smirnov test for concurrency and non-concurrency bugs severity

Shade	Non-concurrency bugs			Concurrency bugs			$|f(x)\text{-}g(x)|$
	Observed Frequency	Observed Proportion	Observed Cumulative Proportion f(x)	Observed Frequency	Observed Proportion	Observed Cumulative Proportion g(x)	
Blocker	41	0.185520362	0.185520362	66	0.298642534	0.298642534	0.113122172
Critical	18	0.081447964	0.266968326	20	0.090497738	0.389140271	0.122171946
Major	118	0.533936652	0.800904977	120	0.542986425	0.932126697	0.131221719
Minor	29	0.131221719	0.932126697	15	0.067873303	1	0.067873303
Trivial	15	0.067873303	1	0	0	1	0
Critical D-value = $D_{221,0.05} = \dfrac{1.36}{\sqrt{221}} = 0.091$				D-value= Sup|f(x)-g(x)| = 0.131221719			

On the other hand, after comparing the different type of concurrency bugs we found that most of the bugs categorized as being part of the *Data race* type in terms of severity belongs to the Major class; the highest population of *Deadlock* bugs belong to Critical class; the highest population of bugs categorized in the *Suspension* type belongs to Critical class; the highest population of bugs corresponding to *Atomicity violation* type belongs to Major and Minor class; the highest population of *Order violation* bugs belongs to Minor class and the highest population of *Starvation* bugs belong to Major class. We can interpret that the *Deadlock* and *Suspension* bugs have higher severity.

Answer RQ3: Concurrency bugs are considered to be more severe than non-concurrency bugs, but the difference is not that large.
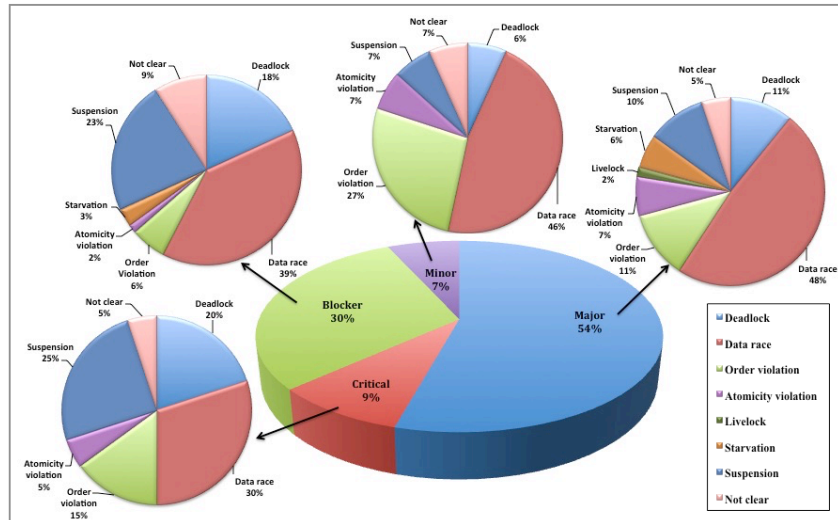

Fig. 5: Concurrency bugs severity

5 Discussion

In our study, we found a much smaller share of concurrency bugs than the one found by other similar studies. This could possibly be due to the different time span of our study and that of other similar studies. An interesting observation is that 70% of the bugs that we found were reported in the five-year interval of 2006-2010, and the remaining 30% were reported in the five-year interval of 2011-2015.

Similarly, the fixing time found by other studies is much larger for concurrency bugs than for non-concurrency bugs. We find a difference, but it is relatively small. This could be due to a large portion of fixing time in other studies relate to reproducing the bugs using the bug scenario in the bug description. In our study we found surprisingly few reports stating difficulties in reproducing the bug.

The involvement of more than one thread cause a concurrency bug. For this reason we predicted to find that concurrency bugs were more severe than non-concurrency bugs. However, we expected most of the "Blocker" bugs to *deadlock* type due to its characteristic and properties but this was not the case. In our study *Data race* was the biggest portion of "Blocker". We can interpreter that *Data race* is the most problematic bug to fix in Hadoop project.

Moreover, our investigation shows that about half of the concurrency bugs are of *Data race* type. The reason could be that *Data race* is more severe than other type of bugs (as shown in Figure 5) and it would be normal if it takes longer time to fix. About 15% of the bugs belongs to the *Suspension* type. By investigating the bug reports' description and comments we noticed that most of the *Suspension* bugs occurred when the developer put a block of code in waiting mode for an unnecessary long time, thereby causing a *Suspension* bug.

5.1 Validity Threats

In the design and execution of this study, there are several considerations that need to be taken into account as they can potentially limit the validity of the results obtained.

- Some concurrency bugs might go unfixed or unreported because they occur infrequently, only on certain platforms/software configurations, or are hard to reproduce. It would be interesting to consider these kinds of bugs, but they are not likely to have detailed discussions. As a result, these bugs are not considered as important as the reported and fixed concurrency bugs that are used in our study. However, based on our investigation , 81 bug reports out of 4872 bug reports tagged as "Cannot reproduce" while 25 of them mentioned at least one of the concurrency keywords (listed in Section 2.2) in their description or comments.
- The reports with other status (i.e., "In Progress" -this issue is being actively worked on at the moment by the assignee- or "Open" -This issue is in the initial 'Open' state, ready for the assignee to start work on it-) were not considered in this study and there is a chance that we did not include the relevant reports.
- Even if the obtained results (for RQ1, RQ2 and RQ3) are based on data samples from a single project, these results might apply to other software as well. More analysis is required to confirm whether this is in fact the case.

6 Related Work

A series of related studies on debugging, predicting and fixing concurrent software have been conducted. In particular, there is a large body of studies on prediction [13, 14, 15, 16] and propagation [17, 18] of bugs in source code.

Most of these studies strive to identify the components or source code files, that are most prone to contain bugs. Fault prediction partially focuses on understanding the behavior of programmers and its effects on software reliability. This work is complementary to the study conducted in this research, which is concentrated on a specific type of bugs (i.e., concurrency bugs) and on understanding their consequences.

In [19], Vandiver et al. analyzed the consequences of bugs for three database systems. This work is focused on presenting a replication architecture, instead of on studying bugs. The authors did not distinguish between concurrency and non-concurrency bugs, and only evaluated whether they caused crash or Byzantine faults.

Three open-source applications bug databases (Apache web server, GNOME desktop environment and MySQL database) are investigated by Chandra and Chen [20], with a slightly different focus than ours. The authors analyzed all types of bugs (only 12 of them were concurrency bugs) to determine the effectiveness of generic recovery techniques in tolerating the bugs. Concurrency bugs are only one possible type of bug that affects their results. In contrast, based on our main objective we focus on a more narrow type of bugs by limiting ourselves to concurrency bugs, but provide a broader analysis (comparing concurrency and non-concurrency bugs) taking into consideration several types of these bugs.

Farchi et al. [21] analyzed concurrency bugs by creating such bugs artificially. They asked programmers to write codes which have concurrency bugs. We believe that artificially creating bugs may not lead to bugs that are representative of the real-world software bugs. We, on the other hand, analyze the bug database of an open-source software, which is well maintained, and widely used software.

Lu et al. examined concurrency bug patterns, manifestation, and fix strategies of 105 randomly selected real-world concurrency bugs from four open-source (MySQL, Apache, Mozilla and OpenOffice) bug databases [1]. Their study concentrated on several aspects of the causes of concurrency bugs, but the study of their effects was limited to determining whether they caused deadlocks or not. We use the same study methodology to find relevant bug reports but we provide a complementary angle by studying the effects of recent concurrency bugs not limited to deadlock and not-deadlock bugs. In other words, according to our objective we used other classification(s) for our study.

7 Conclusion and Future Work

This paper provides a comprehensive study of 4872 fixed bug reports from a widely used open source storage designed for big-data applications. The study covers the fixed bug reports from the last ten years, with the purpose of understanding the differences between concurrency and non-concurrency bugs. Two aspects of these reports are examined: fixing time and severity. Based on a structured selection process, we ended up with 221 concurrency bugs and 221 non-concurrency bugs (sampled). By analyzing these reports we have identified the frequencies of concurrency and non-concurrency bugs. The study also helped us to recognize the most common type of concurrency bugs in terms of severity and fixing time. The main results of this study are: (1) Only a small share of bugs is related to concurrency while the vast majority are non-concurrency bugs. (2) Fixing time for concurrency and non-concurrency bugs is different but this difference is relatively small. (3) Concurrency and non-concurrency bugs are different in terms of severity, while concurrency bugs are more severe than non-concurrency bugs. These findings could help software designers and developers to understand how to address concurrency bugs, estimate the most time-consuming ones, and prioritize them to speed up the debugging and bug-fixing processes.

Acknowledgment

References

1. S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," in *ACM Sigplan Notices*, vol. 43. ACM, 2008, pp. 329–339.
2. R. Peri, "Software development tools for multi-core/parallel programming," in *6th workshop on Parallel and distributed systems: testing, analysis, and debugging*. ACM, 2008, p. 9.
3. W. Zhang, C. Sun, J. Lim, S. Lu, and T. Reps, "Conmem: Detecting crash-triggering concurrency bugs through an effect-oriented approach," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 2, p. 10, 2013.
4. J. Desouza, B. Kuhn, B. R. De Supinski, V. Samofalov, S. Zheltov, and S. Bratanov, "Automated, scalable debugging of MPI programs with Intel Message Checker," in *Proceedings of the second international workshop on Software engineering for high performance computing system applications*. ACM, 2005, pp. 78–82.
5. P. Godefroid and N. Nagappan, "Concurrency at Microsoft: An exploratory survey," in *CAV Workshop on Exploiting Concurrency Efficiently and Correctly*, 2008.
6. M. Süß and C. Leopold, "Common mistakes in OpenMP and how to avoid them," in *OpenMP Shared Memory Parallel Programming*. Springer, 2008, pp. 312–323.
7. P. Fonseca, C. Li, V. Singhal, and R. Rodrigues, "A study of the internal and external effects of concurrency bugs," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*. IEEE, 2010, pp. 221–230.
8. I. Polato, R. R, A. Goldman, and F. Kon, "A comprehensive view of Hadoop research?A systematic literature review," *Journal of Network and Computer Applications*, vol. 46, pp. 1–25, 2014.
9. "What is an Issue - Atlassian Documentation," 2015. [Online]. Available: https://confluence.atlassian.com/jira063/what-is-an-issue-683542485.html
10. S. Abbaspour A., H. Hansson, D. Sundmark, and S. Eldh, "Towards classification of concurrency bugs based on Observable properties," in *1st International Workshop on Complex Faults and Failures in Large Software Systems*, Italy, 2015.
11. S. Qi, N. Otsuki, L. O. Nogueira, A. Muzahid, and J. Torrellas, "Pacman: Tolerating asymmetric data races with unintrusive hardware," in *High Performance Computer Architecture, 2012 IEEE 18th International Symposium on*. IEEE, 2012, pp. 1–12.
12. A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.
13. S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 529–540.
14. N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *27th international conference on Software engineering*. ACM, 2005, pp. 580–586.
15. F. Rahman, S. Khatri, E. T. Barr, and P. Devanbu, "Comparing static bug finders and statistical prediction," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 424–434.
16. C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead, "Does bug prediction support human developers? findings from a google case study," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 372–381.
17. L. Voinea and A. Telea, "How do changes in buggy Mozilla files propagate?" in *Proceedings of the 2006 ACM symposium on Software visualization*. ACM, 2006, pp. 147–148.
18. W.-F. Pan, B. Li, Y.-T. Ma, Y.-Y. Qin, and X.-Y. Zhou, "Measuring structural quality of object-oriented softwares via bug propagation analysis on weighted software networks," *Journal of Computer Science and Technology*, vol. 25, no. 6, pp. 1202–1213, 2010.

19. B. Vandiver, H. Balakrishnan, B. Liskov, and S. Madden, "Tolerating byzantine faults in transaction processing systems using commit barrier scheduling," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 59–72, 2007.
20. S. Chandra and P. M. Chen, "Whither generic recovery from application faults? A fault study using open-source software," in *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*.   IEEE, 2000, pp. 97–106.
21. E. Farchi, Y. Nir, and S. Ur, "Concurrent bug patterns and how to test them," in *Parallel and Distributed Processing Symposium*.   IEEE, 2003, pp. 7–pp.