Does Hardware Configuration and Processor Load Impact Software Fault Observability?

Raza Abbas Syed[1], Brian Robinson[2], Laurie Williams[1]
[1]Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206
{rsyed, lawilli3}@ncsu.edu
[2]ABB Inc., US Corporate Research, Raleigh, NC 27606
brian.p.robinson@us.abb.com

***Abstract*. Intermittent failures and nondeterministic behavior complicate and compromise the effectiveness of software testing and debugging. To increase the observability of software faults, we explore the effect hardware configurations and processor load have on intermittent failures and the nondeterministic behavior of software systems. We conducted a case study on Mozilla Firefox with a selected set of reported field failures. We replicated the conditions that caused the reported failures ten times on each of nine hardware configurations by varying processor speed, memory, hard drive capacity, and processor load. Using several observability tools, we found that hardware configurations that had less processor speed and memory observed more failures than others. Our results also show that by manipulating processor load, we can influence the observability of some faults.**

Keywords-Software testing; observation-based testing; failure observability; empirical study

I. INTRODUCTION

Software maintenance accounts for more than 65% of all costs incurred in the lifecycle of software development [6]. A major component of this cost is debugging and fixing *faults*[1]. Faults may escape to the field as a result of the difficulty in testing large, complex systems adequately before release. A significant amount of software testing research has focused on improving software testing, leading to techniques and tools that aid software engineers in exposing faults. In recent years, advances have been made in test case generation [23, 24], model-based testing [26], regression test selection [25], and graphical user interface (GUI) testing [22]. However, intermittent failures and nondeterministic behavior complicate and compromise the effectiveness of software testing despite these advancements in testing techniques.

A significant challenge to effective and efficient software testing is exposing faults such that they can be observed as failures. The term *observability* refers to the ability to examine a program's behavior, in terms of its outputs, effects on the environment and on other hardware and software components [30]. Over the history of one large product at ABB[2], almost 20% of all reported software failures were determined to be not reproducible by the development and test teams during the product development cycle. For failures detected in test, it is difficult to determine the underlying fault when reproducing the failure is not possible. When the underlying fault cannot be determined, the failure report is often closed and no additional effort is spent on it until additional occurrences are detected. For faults detected in the field, developers spent considerable effort in either remotely debugging the fault or traveling to the customer site itself to determine the fault and fix it.

Nondeterministic behavior in software systems can be attributed to a number of sources. Testers may not have enough visibility or control over all of the inputs to the system under test. For example, the system could perform periodic background tasks that coincide with the test execution. Additionally, different hardware and software configurations can impact the behavior of the system. For example, a failure in a wait queue may not be observable on a fast machine, as messages only enter that queue when the system cannot keep up with incoming requests. Nondeterministic behavior in software is also attributed to concurrency issues (e.g. race conditions) [4]. We limit the scope of this study to only consider the impact of hardware configuration and processor load on fault observability.

The goal of this research is *to improve the observability of software faults by exploring the impact hardware configuration has on intermittent failures and the nondeterministic behavior of software systems*. Specifically, we examine the impact on the observability of software faults due to variation of processor speed, memory, hard drive capacity, and processor load. We conducted two case studies of a set of reported intermittent Mozilla Firefox[3] failures. Ten times per failure, we replicated the conditions that caused the reported failures on nine hardware configurations running Windows XP Service Pack 3 and measured the frequency with which the failure was observed. We also replicated the failure conditions ten times with four different processor loads (0%, 25%, 50%, and 75%).

The rest of the paper is organized as follows. Section 2 presents a summary of related work. Section 3 covers the setup, data collection process, and threats to validity of our study. Section 4 describes the selected failures. Section 5 presents our results and analyses. Finally, Section 6 describes future work and concludes.

[1] A fault is defined as "An incorrect step, process, or data definition in a computer program" [29]

[2] http://www.abb.com

[3] http://www.mozilla.com/en-US/firefox/

II. Related Work

Several studies have appeared in literature concerning nondeterminism [7, 11] as well as observation-based testing [8, 9, 10], but there are few existing studies that have established a correlation between the observability of faults in software and its base hardware. White et al. [1] presented one empirical study carried out on RealPlayer. Their work involved testing RealPlayer on hardware running on different processor speeds, memory, and operating systems. They observed that RealPlayer behaved differently on different configurations, particularly in the way that it failed. They also detected an increased trend of faults with decreasing processor speed and memory. Our study builds on their work using a more rigorous experimental design. In addition, they had no access to the underlying code which limited their ability to study and understand the intermittent nature of the failures.

Duarte et al. [2] present a framework, GridUnit, for testing distributed applications on a grid of multiple heterogeneous environments simultaneously. Their results show that carrying out such tests increases the proof of correctness of software under test by achieving greater environmental coverage for their test suites in a singular or small number of environments. While not completely similar to our research, their work on testing on different environment confirms an important assertion that testing in one environment is not sufficient for verifying that the software executes correctly.

Cohen et al. [13, 14] have discussed the use of combinatorial interaction testing techniques to test different software configurations, for normal as well as regression testing. They have shown that software configurations can have a substantial impact on uncovering faults. Combinatorial testing techniques have been extensively advocated in literature for increasing the efficiency of testing phase by achieving high code coverage while utilizing significantly fewer test cases [15, 18, 19, 20]. Kuhn et al. in [16, 17] found that combinatorial testing is effective for testing software such as Mozilla Firefox. In addition, they found that that 75% of failures in Firefox were dependent on interaction of two or more software configuration parameters. Our work uses combinatorial testing techniques to assess the impact of hardware configuration on the observability of faults.

Porter et al. [27] present a framework for distributed testing of applications called Skoll. The Skoll system focuses on systems with large configuration spaces and distributes testing tasks to user communities worldwide in an effort to use greater computing resources and reduce testing time. They found that their distributed effort found faults sooner than traditional non-distributed techniques would have. Yilmaz et al. [28] found that using covering arrays to select a subset of configuration space for fault characterization allowed for much greater scalability of the Skoll framework.

Problems involving nondeterministic behavior in software have often appeared in literature in conjunction with concurrency problems and parallel execution of programs [3, 4]. Musuvathi et al. [4] at Microsoft Research have developed a tool called CHESS for eliminating nondeterminism in concurrent programs. CHESS explores thread schedules of programs under test in a deterministic manner and uses model-checking techniques to expose any discrepancies in terms of interleaving of events or race conditions. While not directly related to our work, it represents one of the recent research studies undertaken in the area of concurrency testing and intermittent failures.

III. Empirical Study

The goal of our study is *to improve the observability of software faults by exploring the impact hardware configuration has on intermittent failures and the nondeterministic behavior of software systems.* We are interested in the following research questions:

RQ1: Is the observability of a software fault impacted by processor speed? If so, how?

RQ2: Is the observability of a software fault impacted by the amount of memory? If so, how?

RQ3: Is the observability of a software fault impacted by the capacity of the hard drive? If so, how?

RQ4: Is the observability of a software fault impacted by processor load? If so, how?

We address these research questions through a study of selected set of failures from a large open-source application, Mozilla Firefox. Firefox was chosen because it is an active open-source project with a large repository of failures online, many of which have traceability to the code change that were made to fix the underlying faults. The rest of this section describes our procedures for selection of failures, study setup, observed measures, and threats to validity.

A. Step One: Identify Intermittent Failures

Our first step was to collect a set of reported failures with indications that these failures were intermittent and/or non-reproducible. We ran natural language queries on Mozilla's online repository of failures, Bugzilla[4]. We did not restrict our search to any particular version of Firefox. Additionally, no failures related to plug-ins or add-ons were considered, as these represent another factor of configuration outside the scope of this study. The search was, therefore, restricted to Firefox's core codebase. We analyzed hundreds of failure reports returned from out searches, and found 75 failures that matched our criteria. The natural language queries used variations on the keywords such as "timing", "page file", "operating system", "system dependent", "race condition", "deadlock", "concurrency", "pentium", "older machines", and "slow computers".

Once we felt we had gathered a substantial number of intermittent failures, we attempted to reproduce each failure ten times on a machine running Windows XP. The machine used for testing failures had the same specification as reported on Bugzilla. In addition to being able to reproduce a particular failure, we also considered its present status on

[4] http://bugzilla.mozilla.org

Bugzilla, which told us whether the failure had been fixed or not. From our initial set of 75 failures, we selected those that (1) exhibited nondeterministic behavior on our test machine by being intermittently reproducible; and (2) had already been fixed by the Mozilla developers. Using failures that have been fixed assures us that there are identified code changes that can be studied to better understand why the failures were only observable on certain hardware configurations. Using these criteria we narrowed down our sample size to 11 failures. Each failure is described in detail in Section 4 along with an explanation of its code fix.

B. Step Two: Determine Hardware Configurations

We define a set of base hardware configurations on which to replicate our selected failures. These configurations vary in terms of their processor speed, memory, and hard drive capacity. Each of the three factors has multiple levels defined (see Table I). The levels were chosen to best model the range of hardware configurations in use today. Together processor speed, memory, and hard drive capacity constitute the independent variables for our study.

TABLE I. LEVELS OF INDEPENDENT VARIABLES USED IN OUR STUDY

Processor	667Mhz	1Ghz	2Ghz
Memory	128MB	256MB	1GB
HD	2.5GB	10GB	

The application under test was the open-source web browser Mozilla Firefox. The operating system on all configurations was chosen to be Microsoft Windows XP Professional Service Pack 3. While the operating system itself is a potential factor in observability, this study is focusing only on the effect the underlying hardware has on observability. According to Net Applications[5], Windows XP is the most popular operating system in use today with an estimated market share of 71.8%, and the Windows family of operating systems together has 93% of market share. This large market share was the primary motivation in choosing Windows XP for this study.

Hardware configurations were constructed as *virtual machine images* using the freely-available VMWare ESXi Server v4.0[6] software. The three factors of our study were manipulated by using the VMWare vSphere v4.0[7] client application. Although it can be debated how accurately a virtual machine models a real one, we believe such differences are not significant and outside the scope of this study. The ESXi server provides adequate controls for controlling processor speed and memory of a Windows system.

C. Step Three: Hardware Configuration Testing

A full-factorial design of the factors listed in Table I results in 18 total combinations. However, using only pairwise interactions [21], the total number of combinations was reduced by half (see Table II). Pairwise testing enables us to reduce the actual number of test cases by testing only the two-way interaction of variables rather than exhaustively testing all of their possible combinations. Empirical results show that faults are rarely dependent on larger combinations of values, but are more dependent on pairwise interactions of different variables [21]. All of the variables are evenly distributed in the nine combinations below.

The steps needed to invoke the failure were replicated ten times for all selected failures on each of nine hardware configurations, leading to a total of 90 test runs for each failure. Tests were run ten times to capture the frequency a failure was observable on each configuration. The order of testing was randomized to reduce bias.

TABLE II. HARDWARE CONFIGURATIONS USED IN OUR STUDY

No.	Processor Speed	Memory	Hard Disk Capacity
1	667Mhz	128MB	2.5GB
2	667Mhz	256MB	10GB
3	667Mhz	1GB	2.5GB
4	1Ghz	128MB	10GB
5	1Ghz	256MB	2.5GB
6	1Ghz	1GB	10GB
7	2Ghz	128MB	2.5GB
8	2Ghz	256MB	10GB
9	2Ghz	1GB	2.5/10GB

D. Step Four: Processor Load Study

This study involved varying the processor load for all of the selected failures. In addition to the previous study that tested all failures on 0% processor load, we defined three levels of processor load to test - 25%, 50%, and 75%.

The steps needed to invoke the failure were run ten times for each of the selected failures on each of three additional processor load conditions. Since the load generation study only involved manipulating processor load, the failures were tested across a smaller number of configurations that varied in processor speed (see Table III). Including test runs conducted in the previous study, each of the selected failures was run ten times on three configurations with four processor loads making the total number of test runs 120 for each failure.

TABLE III. HARDWARE CONFIGURATIONS USED IN THE LOAD STUDY

No.	Processor Speed	Memory	Hard Disk Size
1	667Mhz	128MB	2.5GB
2	1Ghz	128MB	10GB
3	2Ghz	128MB	2.5GB

E. Baseline

To establish a base of measurement for the observability of failures in Firefox's codebase, a further 12 random failures were selected and tested on the slowest virtual machine configuration i.e., a configuration with processor speed of 667 MHz, 128 MB memory, and 2.5 GB hard drive capacity. The failures chosen for the baseline test all had failure descriptions that indicated the failures were not intermittent.

[5] http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=10
[6] http://www.vmware.com/products/esxi/
[7] http://www.vmware.com/download/vsphere/

F. *Observability Data Collection and Tools*

The data of interest to us in this study included status indicators of the system and the software under test. Using different observability tools listed below, we were able to measure Firefox's memory consumption (working set, page faults per second, and page file size), processor usage, thread count, handle count, and priority base. For the system's performance measures, we measured total processor usage, average disk queue length, and memory pages per second. The sequence of events fired by the application under test was monitored using an event-capturing tool. Finally, we manipulated processor load using a free utility that simulates load for Win9x/2000 systems. All the preceding performance measures constitute the dependent variables for our study.

The tools used for this purpose included:

- Spy++[8] - An event-capturing utility that ships with Microsoft Visual Studio. Helpful in monitoring behind-the-scenes activity of a windows application.
- VMMap[9] - Provides a map of virtual memory for analyzing an application's memory usage.
- Perfmon[10] - A windows system diagnostic utility used for monitoring total processor usage as well as process-specific measurements.
- CPU Grabber[11] – A utility that lets us simulate processor load. Part of Microsoft DirectShow SDK Framework.

G. *Threats to Validity*

Internal validity involves determining effect of other factors that can influence study results without the researcher's knowledge. As this study specifically focuses on nondeterministic behavior of software, any factors that affect the software under test are a risk. This risk was mitigated by using a clean virtual image with only the operating system and Firefox installed. Since testing Firefox involves sending messages on the network, network traffic and other non-controllable factors could make up some of the unexplained variance in the ANOVA models. Also, the tests were run manually. For a few of the failures, this manual timing lead to some unexplained variance in the ANOVA models.

Threats to external validity are conditions that limit the generalization of the results. The primary threats in this study are the use of a single subject program and the use of a small set of failures. Future studies are needed to better understand how prevalent these issues are in Firefox and other types of software.

Threats to construct validity arise when measurement instruments do not properly capture what they are intended to capture. In this study people were used to physically observe the failure, either through directly viewing the deviant behavior or indirectly observing the failure through the data collection tools. Subtle changes in the failure could be missed by the observers.

[8] http://msdn.microsoft.com/en-us/library/aa242713(VS.60).aspx

[9] http://technet.microsoft.com/en-us/sysinternals/dd535533.aspx

[10] http://technet.microsoft.com/en-us/library/bb490957.aspx

[11] http://www.microsoft.com/windows/directx/

IV. SELECTED FAILURES

Based on the criterion mentioned in the preceding section, we arrived at our sample size of 11 failures for in-depth study. This section describes each of those failures in greater detail along with their code fixes.

Failure # 124750: On Firefox version 0.8, and on computers with low processor speeds, this failure resulted in a stealing of focus from the active tab by another tab loading in the background. This failure was timing-related and was evident on web pages that shifted focus to an input field upon loading. For example, loading google.com (which shifts focus to its search box as soon as it loads) and quickly switching to another tab (say yahoo.com) to type a query results in the typed text appearing on the search box in the google tab. This behavior occurred since there was only one focus controller object defined in source code for the entire window. This was part of Firefox's underlying Gecko engine and needed significant modifications. Instead a workaround was put in place to avoid this behavior that simply blocked all focus-stealing calls by checking if the tab requesting focus was also the active tab at that time.

Failure # 200119: This failure occurred when a Certificate Revoke List (CRL) is configured to be automatically updated in Firefox version 1.0. This configuration lead to random crashes of the application when a user attempted to exit. Although this failure was intermittent in our tests initially, it was later found to be always observable upon visiting a particular website. The fix for this failure involved refactoring the code and removing some redundant method calls.

Failure # 264562: This failure involved the "Find As You Type"[12] or incremental find functionality in Firefox. Upon pressing Ctrl-T to open a new tab and pressing some keystrokes in quick succession causes the find bar to appear at the bottom of the window, even though there is a blank page loaded in the browser. This failure was only reported on machines that had low processor speeds. The fix for this failure was to simply block all user input when no web page had been loaded in the browser window.

Failure # 314369: This failure has to do with the difference in minimum marquee speeds between Microsoft Internet Explorer and Firefox. In Internet Explorer, the minimum speed for a marquee is 60ms while in Firefox it is 40ms. This failure could be regarded as a case of always failing, but failing differently (since marquee speeds were different each time the test was run). Some marquees in test cases exhibited inconsistent speeds and their behavior had to be mended in the source code fix to conform to standards similar to that of Internet Explorer.

Failure # 332330: This failure occurred when Ctrl-N and Alt-D were pressed in quick succession of each other. Normally this action would result in shifting of focus to the Firefox address bar but, instead, it triggered the *Open Location* dialog box. . Similar to the failure # 124750, this failure was also observed on systems with low processor

[12] http://www.mozilla.org/access/type-ahead/

speeds. Furthermore, this failure occurred only when a user's homepage was set to *about:blank*. Analyzing the underlying source code, we found that a variable *gURLBar* (representing the address bar) was uninitialized for cases when homepage was set to *about:blank* to improve application performance during startup. The fix was to assign the gURLBar its usual value regardless of a user's homepage setting.

Failure # 363258: This failure was related to the inaccurate millisecond resolution used in earlier versions of Firefox on Windows XP. The millisecond resolution by default was 15 or 16ms which made profiling applications and add-ons for Firefox inaccurate. The fix involved doing a new implementation of the timer with a resolution of 1ms. This was another failure that was deemed deterministic since each test run produced different results i.e., the failure was always observed, just differently each time.

Failure # 380417: This failure was related to the Standard Vector Graphics (SVG) functionality in Firefox. The failure was specifically dependent on timing execution of instructions. The test case for observing this failure had some JavaScript[13] alerts that corresponded to the execution of instructions. The test case had three alerts in all, and one of them corresponded to SVG load event, the appearance of which indicated a normal flow of execution (and hence, the non-observavility of the fault). The fix was to use an event dispatcher to make sure the SVG load event was not ignored and was always captured by a listener.

Failure # 396863: This failure resulted in two sub-menus of the same menu-item being open at the same time. Normally, when focus moves away from one sub-menu, it closes after a set time or earlier if another sub-menu is opened. On a particular build of Firefox containing this failure, the mouse has to be moved quickly from an item in the first sub-menu to the second sub-menu so that both are open at the same time. The fix corrected the behavior of the timer responsible for opening and closing sub-menus.

Failure # 410075: This failure is timing-related and is observed when Ctrl-T and Ctrl-K in quick succession on Firefox's main window. By default, Ctrl-T opens a new tab and Ctrl-K moves focus to the search box at the top-right of the window. When pressed in quick succession, these sequence of keystrokes result in focus moving back to the address bar instead of staying in the search box. The fix to this was to remove a timeout in legacy code that was causing the focus to shift back to the address bar after a specified time interval, and hence, was overriding Ctrl-K.

Failure # 463635: This failure was related with the *All-Tabs* functionality that was introduced in a prototype of an upcoming version of Firefox. The All-Tabs functionality is a new way of switching between tabs inside Firefox, and is modeled after how windows are switched in Windows Vista. In addition, the All-Tabs functionality also includes an incremental search feature. This failure was observed when multiple tabs were open in Firefox and a tab search was run form the All-Tabs window quickly followed by pressing the Enter key. These actions result in focus switching to the presently selected tab in the All-Tabs window rather than the first tab returned in the search results. The fix to this failure involved waiting for the search to be over before shifting focus to the appropriate tab.

Failure # 494116: This failure resulted in audio and video on a system running Microsoft Windows to go out of synchronization. The failure report described the failure as intermittent and that it was observed on normal as well as high processor load. The fix changed the method a Windows system's audio timers were requested to ensure that audio and video were always in synchronization.

[13] http://java.sun.com/

V. Results And Analysis

This section presents the analysis of study observations carried out on different hardware configurations and processor loads.

A. Hardware Configuration Study Results

The hardware configuration study involved replicating the conditions for each failure ten times on each configuration. In summary, of the 11 failures that were under test, the observability of five failures was related to the hardware configuration, and the other six had no such relationship. The ANOVA results for all observed failures are summarized in Table IV.

Figure 1 displays the results for failure # 124750. The figure shows a downward trend in the number of occurrences of this failure as configurations increase in terms of processor speed, memory, and hard drive capacity. The ANOVA results in Table IV suggest a strong dependence on processor speed across all configurations. More than 70% of the observed variation in the data was due to processor speed alone. Memory and hard drive were statistically insignificant with p-values of 0.563 and 0.685 respectively.

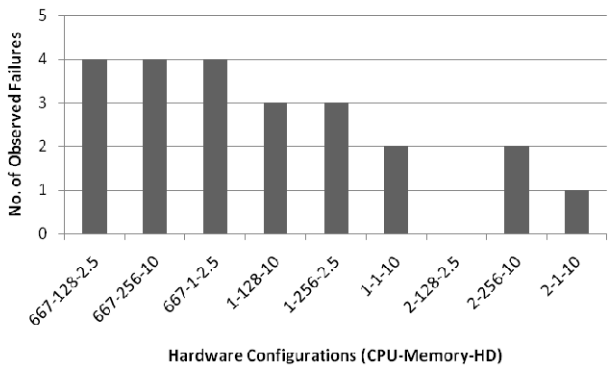

Figure 1. Observed frequency of failure # 124750.

TABLE IV. ANOVA Results for Failure Observability Data

Failure ID	Significant Factor	P-value	Std. Dev.	R-Sq %	R-Sq (adj) %
124750	Processor	0.038	0.745	89.73	72.60
380417	Memory	0.057	0.650	89.41	71.76
396863	Memory	0.004	0.186	98.33	95.54
410075	Processor	0.15	1.773	76.92	38.45

494116	Processor	0.014	0.600	94.58	85.56
264562	Processor	0.037	0.600	90.97	75.93
332330	Processor	0.011	1.100	95.91	89.08

Figure 2 displays the results for failure # 380417. The ANOVA results for this failure suggest a dependence on memory more than processor speed. The shape of the observed behavior suggests a decreased rate of occurrence with increasing memory and processor speed. The rate of occurrence is highest for cases with 128MB memory setting, hence its low p-value in the ANOVA results. Processor speed is less significant than memory in this case, with a p-value of 0.235 and lastly, hard drive is statistically insignificant with a p-value of 0.727.

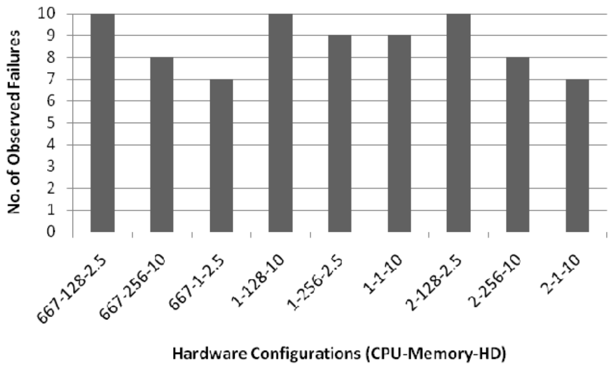

Figure 2. Observed frequency of failure # 380417.

Figure 3 displays the observed behavior of failure # 396863. This failure was most susceptible to manual testing since it involved manually moving the mouse to expose the failure (i.e., opening of two sub-menus at the same time). The figure suggests a regular pattern of occurrence for this bug across all configurations. The ANOVA results show a dependence on all three factors with the most significant being memory with a p-value of 0.004. Processor speed and hard drive were also significant with p-values of 0.029 and 0.052 respectively.

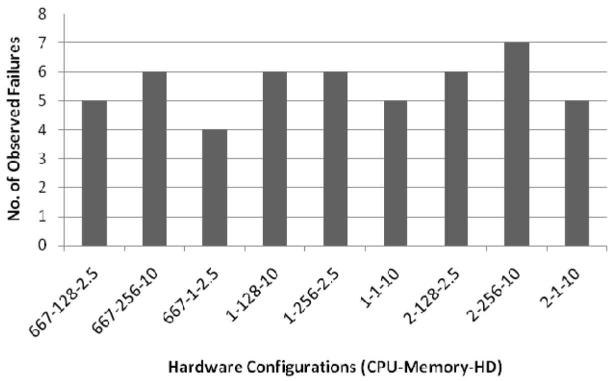

Figure 3. Observed frequency for failure # 396863.

Figure 4 shows the observed behavior for failure # 410075. The ANOVA results suggest a dependence on both processor speed and hard drive factors. The failure description (see Section 4) implies it being dependent only on processor speed, but the ANOVA results show that both hard drive and processor speed have low statistical significance p-values of 0.150 each. Memory is also found to be statistically insignificant with a p-value of 0.453.

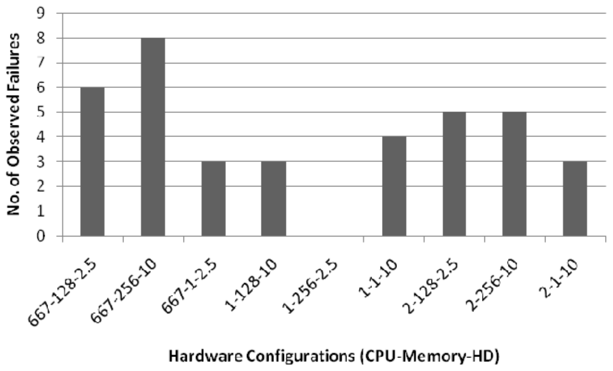

Figure 4. Observed frequency for failure # 410075.

Figure 5 shows the observed behavior for failure # 494116. There is a strong dependence of this failure on processor speed (with a p-value of 0.014) and it was only observable on the first three configurations i.e., configurations that had a processor speed of 667 MHz. These slower configurations were the only ones that had difficulty in playing audio and video smoothly together for the duration of the test. Memory and hard drive were not significant for this failure, with p-values of 0.614 and 0.466 respectively.

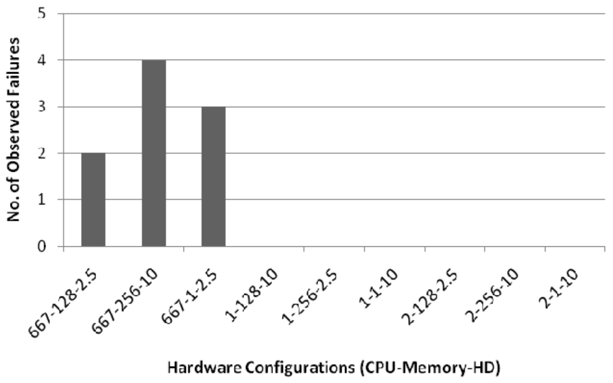

Figure 5. Observed frequency for failure # 494116.

There were six failures whose observability was not related to hardware at all. Failures 200119, 314369, and 363258 were always observable whereas failures 264562, 332330, and 463635 were never observed. Additionally, there were two failures (failures 264562 and 332330) which were not observable in this study but later became observable under high processor load. Their complete load test results are described in the processor load study results.

Figures 6 and 7 show the occurrence rates for these failures respectively.

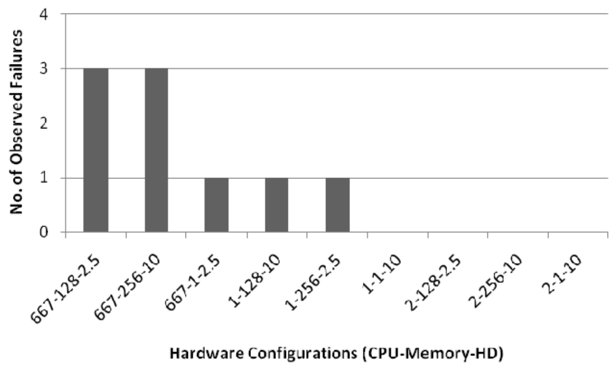

Figure 6. Observed frequency for failure # 264562.

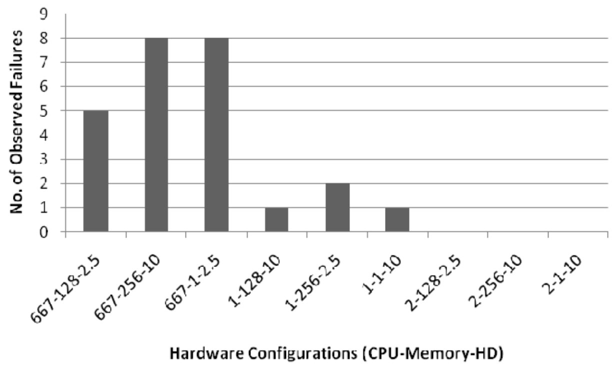

Figure 7. Observed frequency for failure # 332330.

The ANOVA results for both 264562 and 332330 show a strong dependence on processor speed, which accounts for most variation in the data followed by memory. Although these failures are included in the normal load study results, they were never reproducible under normal (or 0%) processor load. For failure # 264562, we simulated processor load by opening 10 different tabs at once (through a drop-down *Live Bookmark* in Firefox) before running the test ten times. For failure # 332330, we simulated the load using CPU Grabber at 25%. Under higher load, both of the failures showed signs of nondeterminism which prompted us to explore this aspect of the failures in greater detail.

In summary, we found that five of the eleven failures selected displayed nondeterministic behavior and were observed intermittently on our test configurations. Our ANOVA results suggest that processor speed and memory capacity were the major contributing factors to such behavior. Except for failure # 410075, all p-values for our intermittent failures were significant (i.e., they were close to or less than 0.05). Additionally, there were several failures that were observed more frequently on the slowest configuration as compared to other configurations with more processor speed and memory. In terms of our research questions, we found that observability of software faults is impacted significantly by both processor speed and memory.

We did not find evidence of hard drive capacity impacting the observability of any failure amongst our selected set of failures.

B. *Case Study 2: Processor Load Generation*

This section describes the results for the processor load study conducted on our selected failures.

Figure 8 shows the results for failure # 124750 when tested under different processor loads. The frequency of the failure increases with increasing load. The ANOVA results of this failure (see Table V) show that more than 80% variance in the data is explained by two factors – processor speed and load. Both are statistically significant with p-values of 0.024 and 0.007 respectively.

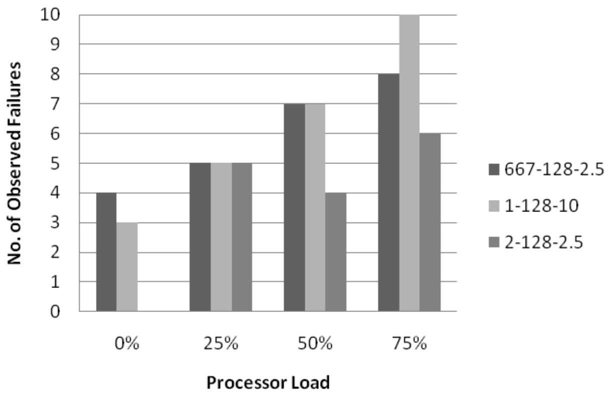

Figure 8. Observed frequency of failure # 124750 under simulated processor load.

TABLE V. ANOVA RESULTS FOR FAILURE OBSERVABILITY DATA USING LOAD GENERATION

Failure ID	Significant Factor	P-value	Std. Dev.	R-Sq %	R-Sq (adj) %
124750	Processor Load	0.007	1.29	89.00	79.84
264562	Processor Speed	0.053	0.64	77.10	58.02
332330	Processor Speed	0.044	2.95	72.79	50.11
380417	Processor Load	0.001	1.11	92.57	86.39
396863	Processor Speed	0.097	1.25	58.55	24.00
410075	Processor Load	0.001	0.86	92.74	86.69
494116	Processor Speed	0.016	0.44	85.86	74.07

Figure 9 shows the observed behavior for failure # 264562 under high processor load. This failure was initially not observable when tested with no processor load. Furthermore, the failure was observed at 25% processor load and higher for the slowest configuration and at 75% processor loads for the other two configurations. Processor load proved to be more effective on the slowest configuration, which is reflected in the ANOVA results. The most significant factor for this failure is processor speed (with a p-value of 0.053) followed by processor load (with a p-value of 0.094).

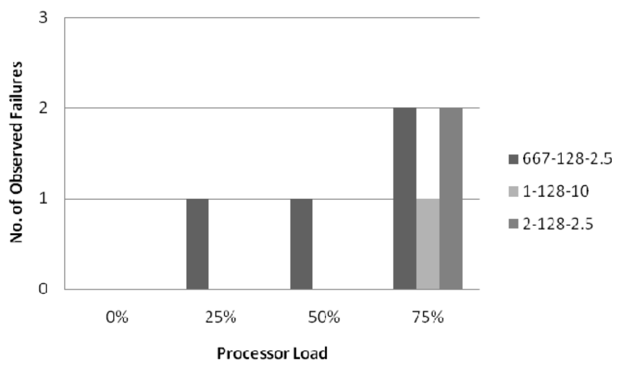

Figure 9. Observed frequency of failure # 264562 under simulated processor load.

Figure 10 shows the observed behavior for failure # 332330 under simulated load. Similar to the previous failure, the failure was observed most on the slowest configuration compared to normal load conditions when it could not be observed at all. This shows a dependence on slower computers and processor load for this failure. The ANOVA results for this bug show a dependence on processor speed more than processor load with p-values of 0.044 and 0.268 respectively.

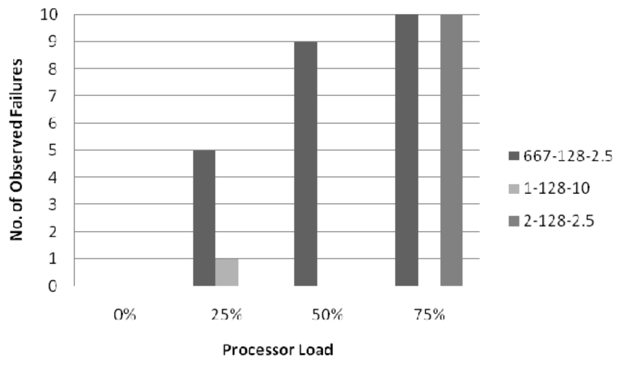

Figure 10. Observed frequency of failure # 332330 under simulated processor load.

Figure 11 shows observed behavior for failure # 380417 under simulated high processor load. The reason we see a downward trend in the figure is because the increased load resulted in the application behaving normally, and thus, we observed the failure lesser number of times. The ANOVA results in Table V confirm a strong dependence on processor load for this failure. The slowest configuration stays relatively consistent across all load tests, but the other two configurations witness a sharp decline in the rate of occurrence of this failure. Of the two factors, processor load is statistically significant with a p-value of 0.001 whereas processor speed is relatively insignificant with a much higher p-value of 0.154.

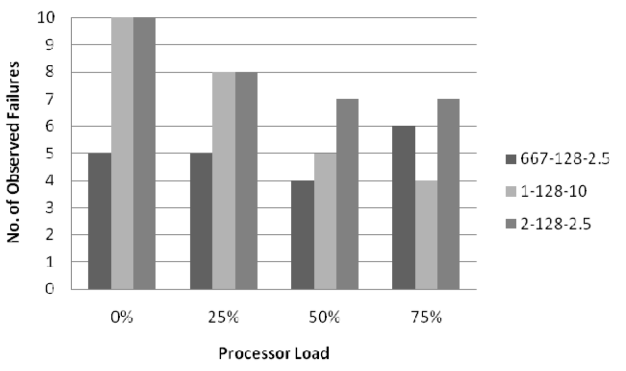

Figure 11. Observed frequency of failure # 380417 under simulated processor load.

Figure 12 shows observed behavior for failure # 396863 under simulated processor load. As mentioned in Section 4, this failure was most prone to the effects of manual testing. The results are similar to the preceding study with the most significant factor being processor speed. The failure occurred due to a malfunctioning timer call (see Section 4 for details), and increased loads did not have an effect on the observability of this failure, expectedly.

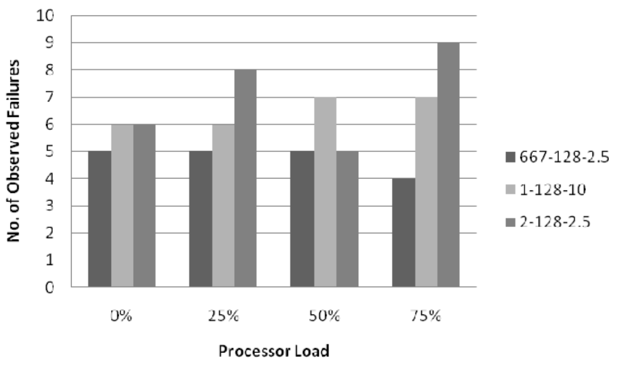

Figure 12. Observed frequency of failure # 396863 under simulated processor load.

Figure 13 shows observed behavior for failure # 410075 under simulated high processor load. For all configurations, the observability of the failure rises sharply to the point that it becomes always observable for 50% load (in some cases) and at 75% load (for all cases). The ANOVA results also show a strong dependence on processor load for this failure (with a p-value of 0.001) even though processor speed was still statistically significant (with a p-value of 0.178).

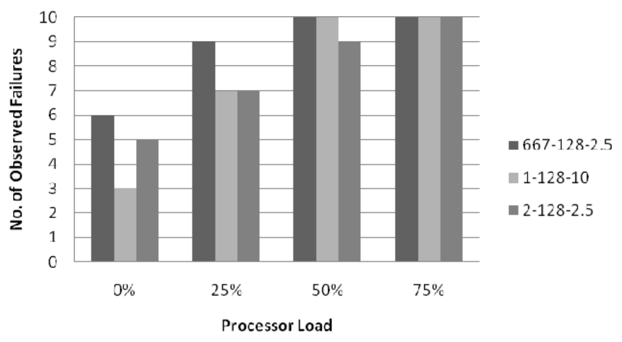

Figure 13. Observed frequency of failure # 410075 under simulated processor load.

Figure 14 shows the observed behavior for failure # 494116 under simulated processor load. This failure was only reproducible on the slowest configurations under normal load (machines with 667 MHz clock speeds), and a similar behavior was witnessed here. There is, however, a downward trend of occurrence with increasing processor load. The ANOVA results show that both processor speed and load were statistically significant, with p-values of 0.016 and 0.029 respectively.

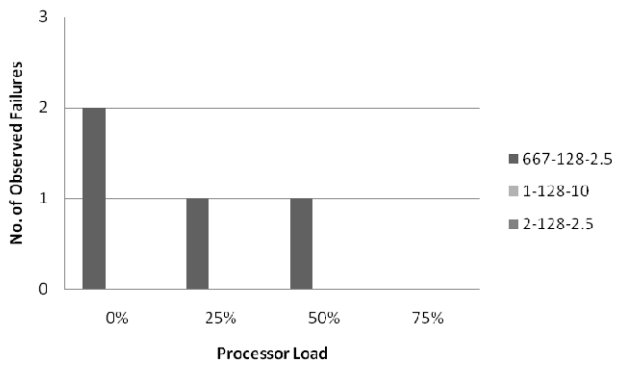

Figure 14. Observed frequency of failure # 494116 under simulated processor load.

In summary, our results indicate that processor load has a significant impact on the observability of software faults. Our ANOVA results suggest both processor speed and load are responsible for exposing nondeterministic behavior of software. Except for failure # 396863, all p-values obtained in our ANOVA results were statistically significant (i.e., they were close to 0.05 level). There were two failures that were not observed on any configuration, but later became observable under higher processor load. Software engineers can, therefore, use our approach to better observe intermittent failures and nondeterministic behavior of software systems.

C. Baseline Failure Analysis

Ten of the twelve baseline failures showed no nondeterministic behavior and were always observable. There were, however, two failures that had differences in their observability - *329892* and *332493*. Both of these failures were related to the then newly-introduced *Places* toolbar functionality in Firefox and occurred in builds released close to each other. The nondeterministic behavior of these failures arose due to inconsistencies in implementation of data structures underlying Places toolbar (specifically, the functionality governing folders and separators). The failures and their fixes were not related to processor speed, memory, or processor load.

VI. Conclusions and Future Work

This paper presents the results of an empirical study to explore the relationship between the observability of software faults as failures and the hardware on which the software under test is running. This study was carried out on an open-source web browser, Mozilla Firefox for Windows XP. We used nine hardware configurations and showed the frequency of the failures on those configurations. We now summarize our results in light of our research questions.

A. *RQ1: Is the observability of a software fault impacted by processor speed? If so, how?*

Our observations indicate that processor speed is a major contributing factor to nondeterministic behavior in software systems, and can be manipulated to increase the observability of software faults. In our hardware configuration study, five of the seven failures that showed nondeterministic behavior depended on the variation in processor speed. Except for one failure, all results were statistically significant with low p-values for processor speed. In addition, our results show that hardware configurations that had less processor speed observed failures more frequently than other configurations with higher processor speeds.

B. *RQ2: Is the observability of a software fault impacted by the amount of memory? If so, how?*

Our results show that memory capacity also influences the observability of software faults, similar to processor speed. Two of the seven failures that displayed nondeterministic behavior in our hardware configuration study were strongly dependent on memory. In both cases, our ANOVA results were statistically significant.

C. *RQ3: Is the observability of a software fault impacted by the size of the hard drive? If so, how?*

Our observations did not indicate that observability of software faults was impacted by hard drive capacity. There was one failure (failure # 410075) where hard drive capacity had the highest p-value of the three factors (0.150), which is not statistically significant.

D. *RQ4: Is the observability of a software fault impacted by processor load? If so, how?*

We conducted a separate case study for observing the effects of processor load on observability of software faults, and found that manipulating processor load increased the observability of all failures under test. In terms of ANOVA results, processor load was statistically significant for four of the seven defects i.e., it had a p-value of less than 0.05 (including cases where processor speed had a lower p-value than load).

In light of our findings, software engineers can increase the observability of faults by testing their systems on configurations that have low processor speeds and memory. In addition, the increased load approach itself can be used as a possible solution to the problem of failure observability, as it increased the frequency of failure detection for failures that were otherwise hard to find.

In future work, we intend to investigate why only some software systems exhibit nondeterministic behavior, and what are the underlying patterns and measures that can be used to predict and, in turn, mitigate such behavior. We also plan to carry out more detailed studies exploring the observability of failures with a larger subset of failures with more factors, and granular configurations.

ACKNOWLEDGMENTS

We would like to thank North Carolina State University Software Engineering Realsearch group for their helpful comments on the paper. This work is sponsored by ABB Corporate Research.

REFERENCES

[1] L.J. White and B. Fei, "Failures of GUI Tests on Different Computer Platforms," *ISSRE 2003 Fast Abstract.*

[2] A. Duarte, G. Wagner, F. Brasileiro and W. Cirne, "Multi-environment software testing on the grid," In *Proceedings of 2006 Workshop on Parallel and Distributed Systems: Testing and Debugging*, pp.61-68, 2006.

[3] Orit Edelstein, Eitan Farchi, Yarden Nir, Gil Ratsaby, and Shmuel Ur, "Multithreaded Java program test generation," IBM Systems Journal, 41(1), 2002.

[4] T. Andrews, S. Qadeer, J. Rehof, S. K. Rajamani, and Y.Xie, "Zing: exploiting program structure for model checking concurrent software," *International Conference on Concurrency Theory*, Sep. 2004

[5] M. Musuvathi, S. Qadeer, "CHESS: Systematic stress testing of concurrent software," In *Logic-Based Program Synthesis and Transformation*, Springer (2006) 15–16 LNCS 4407.

[6] Hans van Vliet. *Software Engineering: Principles and Practice*, 2nd Edition, Wiley, Sept 2000.

[7] Wenbing Zhao, "Byzantine Fault Tolerance for Nondeterministic Applications," *Third IEEE International Symposium on Dependable, Autonomic and Secure Computin.*, pp.108-118, 25-26 Sept. 2007.

[8] D. Leon, A. Podgurski, and L.J. White, "Multivariate visualization in observation-based testing," In *Proceedings of the 22nd International Conference on Software Engineering* (Limerick, Ireland, June 2000), ACM Press, 116-125.

[9] J. Steven, P. Chandra, B. Fleck, and A. Podgurski., "jRapture: a capture/replay tool for observation-based testing," In *Proceedings of the 2000 International Symposium on Software Testing and Analysis* (Portland, Oregon, August 2000).

[10] W. Dickinson, D. Leon, and A. Podgurski, "Pursuing failure: The distribution of program failures in a profile space," In *Proceedings of the Joint 8th European Software Engeneering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engineering*, pages 246–255, Sept. 2001.

[11] J. Slember and P. Narasimhan, "Nondeterminism in ORBs: The Perception and the Reality," *17th International Conference on Database and Expert Systems Applications*, pp.379-384, 2006.

[12] M. Mitchell and J. Jolley, *Research Design Explained,* 4th Edition, New York:Harcourt, 2001.

[13] M. B. Cohen, J. Snyder, and G. Rothermel, "Testing across configurations: Implications for combinatorial testing," *SIGSOFT Softw.Eng.Notes* 31(6), pp. 1-9, 2006.

[14] X. Qu, M. B. Cohen, and G. Rothermel, "Configuration-aware regression testing: An empirical study of sampling and prioritization," In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, pp.75-86, 2008.

[15] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th edition, McGraw Hill, 2005.

[16] D. Kuhn and M. Reilly, "An investigation of the applicability of design of experiments to software testing," In *Proceedings of the NASA/IEEE Software Engineering Workshop*, pp. 91–95, 2002.

[17] D. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, 30(6), pp. 418–421, 2004.

[18] R. Krishnan, S. M. Krishna, and P. S. Nandhan, "Combinatorial testing: Learnings from our experience," *SIGSOFT Software Engineering Notes*, 32(3), pp. 1-8, 2007.

[19] K. Burr and W. Young, "Combinatorial test techniques: Table-based automation, test generation and code coverage," In *Proceedings of the International Conference on Software Testing Analysis & Review*, 1998.

[20] D. Hoskins, R. C. Turban, and C. J. Colbourn, "Experimental designs in software engineering: D-optimal designs and covering arrays," In *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*, pp. 55-66, 2004.

[21] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "Combinatorial Design Approach to Test Generation", IEEE Software, pp. 83-88, 1996.

[22] Atif M. Memon, "An event-flow model of gui-based applications for testing," *Software Testing, Verification and Reliability*, vol.17, no.3, pp.137-157, 2007.

[23] N.Tillman and J. de Halleux, "Pex - white box test generation for .NET," In *Proceedings of the International Conference on Tests and Proofs*, pp.134-153, 2008.

[24] C. Csallner, N. Tillmann, and Y. Smaragdakis, "DySy: Dynamic symbolic execution for invariant inference", In *Proceedings of the 30th international conference on Software engineering*, pp.281-290, 2008.

[25] X. Qu, M. B. Cohen, and G. Rothermel, "Configuration-aware regression testing: An empirical study of sampling and prioritization," In *Proceedings of the International Symposium on Software Testing and Analysis*, 2008.

[26] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: A systematic review," In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*.

[27] A. Memon, A. Porter, C. Yilmaz, A. Nagarajan, D. Schmidt, and B. Natarajan, "Skoll: distributed continuous quality assurance," In *Proceedings of 26th International Conference on Software Engineering, 2004*, pp. 459-468, May 2004.

[28] C. Yilmaz, M.B. Cohen, and A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," In *Intl. Symp. on Software Testing and Analysis (ISSTA)*, pp. 45-54, July 2004.

[29] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, 1990.

[30] P. Amman and J. Offut, *Introduction to Software Testing*, Cambridge University Press, 2008.