Why Software Hangs and What Can Be Done With It *

Xiang Song, Haibo Chen and Binyu Zang

Parallel Processing Institute, Fudan University
{*xiangsong, hbchen, byzang*}*@fudan.edu.cn*

Abstract

Software hang is an annoying behavior and forms a major threat to the dependability of many software systems. To avoid software hang at the design phase or fix it in production runs, it is desirable to understand its characteristics. Unfortunately, to our knowledge, there is currently no comprehensive study on why software hangs and how to deal with it. In this paper, we study the reported hang-related bugs of four typical open-source software applications, aiming to gain insight into characteristics of software hang and provide some guidelines to fix them at the first place or remedy them in production runs.

1 Introduction

Software dependability is crucial to server and user applications, especially mission-critical ones. Unfortunately, *software hang*, a phenomenon of unresponsiveness, is still a major threat to software dependability. This kind of bug exists in many commodity software systems such as web browsers, database servers and office applications. Most hang bugs do not manifest noticeable effect during normal execution. However, when they emerge, users are unable to get response within expected time. Even worse, a serious hang bug may freeze the whole system and mandate a reboot.

Most previous software bug studies focused on specific systems such as operating system errors [1] and network service bugs [9]. But none of them focused on software hang bugs. Though there are several studies on a particular cause of possible software hang such as deadlock bugs [3, 4], infinite loop and I/O blocking bugs [13, 10] and concurrency [5], it is still unclear whether they are major causes and how they contribute to software hang.

*This work was funded by China National Science Foundation under grant numbered 90818015, Shanghai Leading Academic Discipline Project (Project Number: B114) and the Chun-Tsung Undergraduate Research Endowment.

In this paper, we present the first comprehensive study on real-world software hang bugs. We study the reports of hang-related bugs from four popular open source applications: MySQL, PostGre, Apache HTTPD server and Firefox, which are widely used in three-tier browser-server architecture. In total, we collect 307 confirmed hang bugs and analyze the characteristics of 233 bugs with known root causes (categorized as *Certain*). We also examine the rest 74 bugs (categorized as *Uncertain*) to study the obstacles to fix these bugs.

Our study makes the following observations:

- Applications not only heavily suffer from well-known hang causes such as deadlock, data race and infinite loop, but also from *design errors* and *execution environments*.
- Uncertainty of the root cause of a hang bug is the main obstacle to fix the bug, while concise test cases for bug reproduction could significantly accelerate the progress of bug fixes.
- The fix time span study reveals that environment related bugs, infinite loop bugs and concurrency bugs survive much longer than other types of bugs.
- To mitigate hang-related bugs, operators should check the protocol consistency before applying software update, check the runtime resource before running an application and simulate the runtime environment before deploying the application.

The rest of the paper is organized as follows. Section 2 presents the bug survey methodology of our study. A full analysis of hang bugs in the *Certain* category is presented in Section 3. The study on how bugs are fixed is presented in Section 4. Section 5 discusses our observations and suggestions on avoiding hang bugs. We survey related work in Section 6, and conclude our work in Section 7.

2 Bug Survey Methodology

2.1 Hang Bug Sources

We choose three server applications and one client application for our study: *MySQL*, *PostGre*, *Apache HTTPD server* and *Firefox*. They are widely used in three-tier browser-server architecture with well-maintained bug databases. Concurrency is widely used in these applications to handle multiple requests. We believe the characteristics of hang bugs of these applications should be representative for many other applications.

2.2 Survey Methodology

We collect hang bugs from the bug databases of these four applications. In order to collect hang-related bugs, we use a set of keywords to process the search, for example, '*hang*', '*freeze*' and '*unresponsive*'. After collecting the reports, we manually analyze each bug report and make sure whether it is a hang bug or not and find out the root cause of each by studying the comments, test cases, patches, and source code. We also managed to reproduce several bugs to analyze their characteristics. In total, we studied 307 confirmed hang bugs.

In general, we divide these collected hang bugs into two categories, *Certain* and *Uncertain*, representing whether the root cause can be found or not. We get 233 *Certain* bugs and 74 *Uncertain* ones. The details are shown in Table 1. As for the hang bugs belonging to the *Uncertain* category, we carefully confirm that they are true hang, by either finding confirmations from application users and maintainers or manually reproducing them.

Application	Description	Certain	Uncertain
MySQL	Database	97	39
PostGre	Database	61	15
Apache	Web server	35	8
Firefox	Web browser	40	12
Total		233	74

Table 1: Distribution of Certain & Uncertain hang bugs

3 Bug Analysis

For hang bugs in the *Certain* category, we divide them into nine sub-categories according to their root causes. A general overview of the categorization is shown in Table 2. We describe each sub-category and discuss several representative hang bugs in this section.

3.1 Bug Categorization

Configuration One application may run across various platforms under different configurations. Carelessly configured software may encounter serious hang problems. For example, a mis-configured port number might fail a connection and make a running MySQL cluster hang.

Reason	#Bugs	Percentage
Configuration	13	5.58%
Design	37	15.88%
Environment	39	16.74%
Infinite Loop	32	13.7%
Inefficient Algorithm	14	6.01%
Concurrency	54	23.2%
User Operation Error	20	8.58%
Plug In	12	5.15%
Others	12	5.15%
Total	233	100%

Table 2: Bug distribution of nine sub-categories.

Design A large fraction of hang bugs (15.88%) is related to design problems. They are caused by application design errors, maintenance mistakes or unexpected working scenarios. A detailed discussion is presented in section 3.2.1.

Environment A set of hang bugs are caused by unexpected environments applications depending on, composing more than 16% of the *Certain* category. We will discuss it in detail in section 3.2.2.

Infinite Loop In this case, the loop termination condition cannot be satisfied and the CPU resource is exhausted. In total such bugs account for 13.7% of all *Certain* hang bugs. As this is another important category, we will discuss it in detail in section 3.2.3.

Inefficient Algorithm This kind of problem is not related to correctness issues. However, an ill-designed algorithm will be exponentially slower than a well-designed one. When the ill-designed one is applied, the related operations would consume most of CPU and memory resources for a long period, appearing like running in an infinite loop.

Concurrency Modern applications are usually multi-threaded or multi-processed in order to exploit the abundant resources in modern many-core architectures. Concurrency-related bugs are caused by wrong synchronization such as data races, deadlocks and live locks, which consist of 23.2% of all hang bugs in the *Certain* category. We will discuss them in detail in section 3.2.4.

User Operation Error Hang bugs in this case are caused by operational mistakes from users. Misunderstanding or misusage of operations is the main cause of the problem. For example, forgetting to commit an in-flight transaction will hang the related ones.

Plug-In All Plug-In & Extension caused hang bugs are spotted in Firefox bugzilla. There are several kinds of plug-in & extensions such as Flash, Adblock, and Adobe Acrobat Reader. Debugging this kind of bugs requires the cooperation of customers, application maintainers and plug-in & extension providers. This could increase the complexity of

the bug-fix progress.

Others There are approximately 5% of other four kinds of hang bugs: *Resource Exhaustion*, *Resource Leakage*, *Programming Error* and *Internal Structure Corruption*.

3.2 Critical Hang Bug Analysis

About 70% of collected *Certain* hang bugs belong to *Design*, *Environment*, *Infinite Loop* and *Concurrency* categories. In this section, we will study the main characteristics of each category and provide several case studies.

3.2.1 Design

Design category contributes to 15.88% of total hang bugs. We further divide them into two sub-categories: *Design & Protocol* and *Unexpected Input*.

Design & Protocol During the program design and maintenance progress, some program fragments will be added, modified or removed to fix bugs, add new features or improve performance. The internal protocol may be changed or even broken, which will cause serious hang bugs. Unfortunately, such side effects are usually hard to detect and correct. The most common case is that one component fails to send the required message to another, hanging the waiter.

Figure 1 shows an example bug related to protocol broken. In MySQL, all arguments to its internal function *NAME_CONST* should be consistent expressions. A software evolution, however, breaks such an invariant, causing the client hang for responses from the server.

Application: MySQL 5.0.40 Platform: FreeBSD
Description: MySQL's internal function NAME_CONST requires all of its arguments to be constant expressions. This constraint is checked in the Item_name_const::fix_fields method. Yet if the argument of the function is not a constant expression no error message will be reported to the client end. As a result, the client will hang forever waiting for a response from the server end.

Figure 1: Typical protocol design hang bug

Unexpected input Though many software systems usually have many test cases to verify expected inputs, there is still coverage problems that leaves some legal but untested inputs from abnormal scenarios, which could cause a program to enter an undefined state such as software hang. This kind of bug is hard to detect but relative easy to resolve when the erroneous input is determined.

3.2.2 Environment

Environment category takes up to 16.74% of hang bugs. We group several of them into more specific sub-categories according to the system components they depend on. Table 3 lists such categorization and their distributions.

I/O disconnection A manually plugging-out of network cable, a shutdown of network services, a processing error of hub and other accidents would cause this kind of bugs.

Module Dependency Modules is a typical way to provide convenient extensions to the main functionality of software. However, this also increases the possibility of software hangs if the interactions between modules and the main software are not well designed or the modules are buggy.

Multi-threading Support This category concerns about the multi-threading support of the underling environment. A well-known example is the *thread_rwlock_unlock* assembly code bug in glibc. We discover it in MySQL hang bug report, while there are many other similar bug reports as those in Redhat, Ubuntu and others.

Resource Unavailable *Resource unavailable* is another important source of environment related hang bugs. Examples include a missing daemon causes the whole Apache HTTPD server to hang and some missing files hang the initialization progress of a PostGre Admin process.

Type	Bugs	Percentage
I/O disconnect	5	12.82%
Module Dependency	3	7.69%
Multi-thread support	3	7.69%
Resource unavailable	7	17.95%
Others	21	53.85%
Total	39	100%

Table 3: Distribution of environment related hang bugs

However, the rest 53.85% cannot be labeled into any of the above sub-categories. Most of them relate to the entire execution environment as listed below.

- Operating System.
- Simulation environment, like Cygwin for PostGre.
- Zombie applications running in the background.
- Network environment, especially for multi-level http servers.
- Firewall.

3.2.3 Infinite Loop

Infinite Loop category takes up to 13.7% of hang bugs. We divide them into three sub-categories.

Unsatisfied loop condition As the name suggested, this kind of loop is caused by an infinite loop like "while (true)". The problem is that the loop termination condition cannot be satisfied, exhausting the CPU resources.

Cycle-linked list A careless processing of a linked list may cause an infinite loop. When part of the list nodes form a cycle and the target node is outside of the cycle, a walk through this cycle towards the target node will run infinitely. A typical example is in Figure 2.

Application: MySQL 5.0 Platform: ALL
Description: In mysql_make_view for joining algorithm views, views' sub-queries are inserted into select_lex->slave(->next)* chain. In case a join has several views, adding the same sub-queries several times will form a loop on the above chain which breaks many parts of the data.

Figure 2: Example of cycle-linked list caused hang

Others Other kinds of infinite loop hang bug are similar to the above two. A cycle is always formed. An example is a hang caused by a self-referenced bookmark reported in Firefox bugzilla.

3.2.4 Concurrency

This category takes up to 23.2% of hang bugs. Modern applications are usually multi-threaded or multi-processed in order to run efficiently in modern many-core architecture. As more CPU cores are used, hang bugs due to concurrency are becoming more and more serious. A summary of different sub-categories is shown in Table 4.

Type	Deadlock	Live lock	Race	Total
Bugs	32	13	13	58
Percentage	59.26%	24.07%	16.67%	100%

Table 4: Distribution of concurrency related hang bugs

Deadlock There are many examples of deadlock related hang, especially in database applications (MySQL and PostGre) during modifying tables or accessing data. Although some deadlock detection mechanisms emerged in the past few years, it still exists and takes significant part of software hang.

Live lock Live lock is generally not caused by locking situations such as deadlock. Some even do not need a single lock to generate. There are various causes of live lock:

- Application waiting for a none-existing thread or object, which is not created or already cleaned.
- Application waiting for a message which has already been sent due to an error state on the other side.
- A full-filled log file causing a block of log operations.
- A signal notification error because of privilege lacking.

Race When multi-threading or multi-processing programming model is used, races between different threads or processes are common. Race related hangs are usually caused by a wrong sequence of status transferring, such as event delivering sequence and state changing sequence. Replay is a good method to deal with race problems. However, unlike deadlock, *Race* bug is more complex for the following reasons:

- Status variables are not as apparent as locks.
- Combination of status is more complex than locks.
- There are multiple forms of status.

4 Bug Fix Study

In this section, we first provide the study on bug fix strategies. Then we examine the *Uncertain* category to show that the uncertainty of root causes is a main obstacle to hang bug fix. Finally, we present the study on the fix time span of each kind of bugs.

4.1 Fix Strategy

There are several bug fix strategies to deal with hang bugs during our study.

- **Fix by patch** A well-tested patch is applied.
- **Bypass error code** Error code is removed or the error condition is filtered. However, this may sacrifice scalability or functionality of an application.
- **Unfixable** Some bugs cannot be fixed or even bypassed, because of the complexity of the bug and the importance of its functionality. The only thing can do is to notify users the existence of certain bug.
- **Update** The last strategy is updating the application with a new design or implementation. Usually when root cause is not discovered, trying a newer version and expecting a fix is the only choice.

Category	Bugs	With reproducing description	Percentage
Uncertain no fix	33	12	36.4%
Uncertain quick fix	14	12	85.7%
Uncertain patch fix	10	9	90%
Uncertain version fix	17	8	47.1%
Total	74	41	55.4%

Table 5: Distribution of *Uncertain* category

4.2 Obstacles to Bug fix

One major obstacle to hang bug fix is the uncertainty of how a bug occurs, where the bug locates and what causes the bug. As shown in Table 5, nearly a half of them are not fixed as bug reports indicate, while only 13.5% were fixed by patches. **Patch fix & quick fix** means the bug is fixed by patch or will be fixed in the next version sooner. Though we failed to figure out their causes, maintainers might have discovered the root causes of them. With 21 out of 24 bug reports, we successfully find the test cases to reproduce each bug. **Version fix** means the hang bugs are fixed by version update. With 8 out of 11 bug reports with reproducing descriptions, maintainers quickly responded with which version had fixed or would fix the bug. While among the remaining 6 bugs, 5 are fixed by trying the newer versions repeatedly. **No Fix** means no cause of hang is found and no fix report is found in bug reports. They only contain conform messages and scenario descriptions of hang. Only 12 bug reports provide steps to reproduce the bugs, most of which are useless.

Since uncertainty is an important obstacle to hang bug fix, concise test cases for bug reproduction are the keys to accelerate the bug-fix progress.

4.3 Fix Time Span

We checked the fix time span of 160 hang bugs, all of which belong to the *Certain* category. We did not analyze the time span of hang bugs from PostGre because its bug report time is not well logged. We also did not check the time span of bugs from *Plug-in & Extensions* category, as they are usually not maintained by the application providers. Table 6 shows the average fix time span of eight *Certain* subcategories.

There are three categories of hang bugs requiring more than 100 days on average to fix, *Environment*, *Infinite Loop* and *Concurrency*. Especially, the *Race* and *Pipe* (part of *Live Lock*) related hang bugs, which belong to the *Concurrency* category, take a much longer time to fix (213 days and 444 days respectively). Three of eight analyzed bugs in the *Race* category even take more than 1.5 year to fix as they are very complex to find the root causes.

Reason	Days
Configuration	74.10
Design	74.32
Environment	147.46
Infinite Loop	140.23
Inefficient Algorithm	96.50
Concurrency	137.47
User Operation Error	85.64
Others	62.78

Table 6: The average fix time span of eight hang bug categories. (We ignored two bugs requiring about 2000 days to fix)

5 Lessons Learnt from Study

In this section, we present our observations and suggestions on how to improve hang bug fix progress.

5.1 Bug Distribution within Applications

Different applications usually have different usages, designs and implementation strategies. To gain the software hang information about each application, we present a generic summary in Table 7. From the table, we observe that the four major hang bug contributors affect any of the four applications heavily. However, they also suffer significantly from specific kind of hang bugs as well:

- Database applications suffer heavily from Concurrency problems, which comprise about 25 percents of all bugs.
- Environment related hang bugs take up nearly one third of all in *Apache HTTPD server*.
- Firefox, the web browser is affected by inefficient algorithm heavily.

Reason	MySQL	PostGre	HTTPD	Firefox
Configuration	3.09%	4.92%	11.43%	10.71%
Design	21.65%	14.75%	11.43%	10.71%
Environment	13.40%	18.03%	34.29%	10.71%
Infinite Loop	12.37%	14.75%	14.29%	21.43%
Inefficient Algm	6.01%	1.64%	0%	21.43%
Concurrency	23.2%	26.23%	14.29%	17.86%
User Oper Error	7.22%	16.39%	8.57%	0%
Others	6.19%	3.28%	5.71%	7.14%
Total	100%	100%	100%	100%

Table 7: Hang bug distribution

5.2 Observations

We get several interesting observations about hang bug causes during our study.

1. Many hang bugs, especially belonging to *Infinite Loop*, *Design* and *Race*, are actually related to *state inconsistency*. How to maintain the right state at the right time is a crucial problem to programmers. This also raises challenges to researchers to provide practical and easy-to-use tools.
2. Incompatibility introduced by TCP stack change and program update may cause hang problems as well as other bugs. Some of the hang bugs treated as *Design* bugs are actually caused by internal compatibility problem between different components such as the request processing incompatibility. As modern software becomes more and more complex and heavily relies on cooperation between components, the compatibility problem becomes more and more serious.
3. *Resource Unavailable* is a major cause of hang, as discussed in section 3.2.2 and section 3.2.4, no matter what is the root cause of the unavailability. Checking the availability of runtime resource before execution should be a common program rule during software development.
4. *Configuration* problems might cause software hang. This requires programmers to write better auto-configuration code to adjust the software according to the environments.
5. Users with diverse knowledge may have different understanding of how software works. It is important to tune the software and to inform users how to correctly use software. Documents should have more details and highlights on the critical parts of using software.
6. As mentioned in section 3.2.3, *linked list* is vulnerable to hang. As it is widely used in modern software, it would be good to use well-encapsulated interface to manipulate linked list to avoid software hang.

5.3 Methodologies to Improve Bug Fix

The followings are suggestions on detecting software hang and improving hang bug fix based on our study:

1. Incompatibility of communication protocols between different components may be introduced by software updates. Hence, a general protocol consistency checker should be provided and applied for software updates.
2. It is important for developers to provide enough test cases with different execution *environments*, to mitigate hang bugs caused by unexpected environments.
3. *Runtime checking* or *resource insurance mechanism*, such as checking the underlining TCP stack to avoid firewall incurred stack inconsistency problem, can help to avoid resource related problems.
4. Configuration scripts should be designed to satisfy as many hardware and software combinations as possible, and auto configuration tools should adjust themselves when the execution environment changes. A simulation on different combination of various hardware and software can help to detect script errors earlier before deploying the application.

6 Related Work

Bug characteristic study: There has already been a lot of work on studying the other types of bugs in commercial software, including Operating System errors [1], system utilities [6], network applications [9] and Concurrency bugs [5]. Compared to these studies, our study is unique in providing a comprehensive study on the characteristics of software hang.

Bug analysis and debugging tools: Lots of studies have been done to analyzing and detecting bugs. Replay techniques [7, 11] is heavily discussed in the past ten years. Some remote logging techniques [12] are developed to enable bug analysis in distributed systems and the internet. Recent work also proposed to statically check semantic consistency of software [2]. These techniques can be used on debugging some of hang bugs in categories like *Concurrency*, *Infinite Loop* and *Design*.

Hang detecting tools: There are several techniques exist to detect software hang [8] with the help of hardware or using kernel modules. Most of such approaches use a heartbeat technique, which works like a watchdog helping detecting the hang of the system. However, the satisfied timeout is very hard to decide [10] in practice.

7 Conclusion

Software hang is a severe threat to software dependability. In this paper, we present a comprehensive study on the characteristics of hang-related software bugs from four popular open-source applications. Our study presents nine categories of bugs that are major causes of software hang. Design, Environment, Infinite Loop and Concurrency are four main contributors of software hang. The bug fix study shows that a well-formed bug report is a key to accelerate hang bug fix progress. We also provide several observations and suggestions on how to improve hang bug fix progress.

References

[1] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An empirical study of operating systems errors. In *Proc. SOSP*, pages 73–88, 2001.

[2] I. Dillig, T. Dillig, and A. Aiken. Static error detection using semantic inconsistency inference. In *Proc. PLDI*, pages 435–445, 2007.

[3] D. Engler and K. Ashcraft. RacerX: effective, static detection of race conditions and deadlocks. *ACM SIGOPS Operating Systems Review*, 37(5):237–252, 2003.

[4] T. Li, C. Ellis, A. Lebeck, and D. Sorin. Pulse: a dynamic deadlock detection mechanism using speculative execution. In *Proc. USENIX ATC*, pages 3–3, 2005.

[5] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. *ASPLOS*, 2008.

[6] B. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[7] P. Montesinos, L. Ceze, and J. Torrellas. DeLorean: Recording and Deterministically Replaying Shared-Memory Multiprocessor Execution Efficiently. In *Proc. ISCA*, pages 289–300, 2008.

[8] N. Nakka, G. Saggese, Z. Kalbarczyk, and R. Iyer. An architectural framework for detecting process hangs/crashes. In *European Dependable Computing Conference (EDCC)*, 2005.

[9] D. Oppenheimer. Why do Internet services fail, and what can be done about it? In *Proc. USITS*, 2003.

[10] S. Peter, A. Baumann, T. Roscoe, P. Barham, and R. Isaacs. 30 seconds is not enough!: a study of operating system timer usage. In *Proc. Eurosys*, pages 205–218, 2008.

[11] S. Srinivasan, S. Kandula, C. Andrews, and Y. Zhou. Flashback: a lightweight extension for rollback and deterministic replay for software debugging. In *Proc. USENIX ATC*, pages 3–3, 2004.

[12] C. Verbowski, E. Kiciman, A. Kumar, B. Daniels, S. Lu, J. Lee, Y. Wang, and R. Roussev. Flight data recorder: monitoring persistent-state interactions to improve systems management. In *Proc. OSDI*, pages 117–130, 2006.

[13] X. Wang, Z. Guo, X. Liu, Z. Xu, H. Lin, X. Wang, and Z. Zhang. Hang analysis: Fighting responsiveness bugs. In *Proc. Eurosys*, 2008.