# Windows XP Kernel Crash Analysis

*Archana Ganapathi, Viji Ganapathi, and David Patterson*
– University of California, Berkeley

## ABSTRACT

PC users have started viewing crashes as a fact of life rather than a problem. To improve operating system dependability, systems designers and programmers must analyze and understand failure data. In this paper, we analyze Windows XP kernel crash data collected from a population of volunteers who contribute to the Berkeley Open Infrastructure for Network Computing (BOINC) project. We found that OS crashes are predominantly caused by poorly-written device driver code. Users as well as product developers will benefit from understanding the crash behaviors elaborated in this paper.

## Introduction

Personal Computer (PC) reliability has become a rapidly growing concern both for computer users as well as product developers. Personal computers running the Microsoft Windows operating system are often considered overly complex and difficult to manage. As modern operating systems serve as a confluence of a variety of hardware and software components, it is difficult to pinpoint unreliable components.

Such unconstrained flexibility allows complex, unanticipated, and unsafe interactions that result in an unstable environment often frustrating the user. To troubleshoot recurring problems, it is beneficial to data-mine, analyze and document every interaction for erroneous behaviors. Such failure data provides insight into how computer systems behave under varied hardware and software configurations.

To improve dependability, systems designers and programmers must understand operating system failure data. In this paper, we analyze crash data from a small number of Windows machines. We collected our data from a population of volunteers who contribute to the Berkeley Open Infrastructure for Network Computing (BOINC) project. As our analysis is based on a small amount of data (with a self-selection bias due to the nature of BOINC), we acknowledge that our results do not represent the entire PC population. Nonetheless, the data reveals several useful results for PC users as well as researchers and product developers.

Most Windows users have experienced at least one "bluescreen" during the lifetime of their machine. A sophisticated PC user will accept Windows crashes as a fact and attempt to cope with them. However, a novice user will be terrified by the implications of a crash and will continue to be preoccupied with the thought of causing severe damage to the computer. Analyzing failure data can help users gauge the dependability of various products and understand the source of their crashes.

From a research perspective, the motivation behind failure data-mining is manifold. First, it reveals the dominant failure cause of popular computer systems. In particular, it identifies products that cause the most user frustration, thus facilitating our efforts to build stable, resilient systems. Furthermore, it enables product evaluation and development of benchmarks that rank product quality. These benchmarks can influence design prototypes for reliable systems.

Within an organization, analyzing failure data can improve quality of service. Often, corporations collect failure data to evaluate causes of downtime. In addition, they perform cost-benefit analysis to improve service availability. Some companies extend their analyses to client sites by gathering failure data at deployment locations.

For example, Microsoft Corporation collects crash data for their Windows operating system as well as applications used by their customers. Unfortunately, due to legal concerns, corporations such as Microsoft will usually not share their data with academic research groups. Companies do not wish to reveal their internal vulnerabilities, nor can they share third party products' potential weaknesses. In addition, many companies disable the reporting feature after viewing proprietary data in the report. While abundant failure data is generated on a daily basis, very little is readily sharable with the research community.

The remainder of this paper describes our data collection and analysis methodology, including: related work in the areas of system dependability and failure data analysis, background information about Windows crash data and the data collection process, crash data analysis and results, a discussion of the merits of potential extensions to our work, and a conclusion.

## Related Work

Jim Gray's work [Gra86, Gra90] serves a model for most contemporary failure analysis work. Gray did not perform root cause analysis but rather Outage Cause that considers the last in the fault chain. In 1989, he found that the major source of outages was software, contributing about 55%, far outrunning its immediate successor, system operations, which contributed 15%. This observation led him to blame software for almost every failure. In an earlier study [G05, GP05], we analyzed Windows application crashes to understand causal relationships in the user-level. Departing from Gray's outage cause analysis, in our study we perform root cause analysis under the assumption that the first crash in a sequence of crashes is responsible for all subsequent crashes within that event chain.

The past two decades have produced several studies in root-cause analysis for operating systems (OS) ranging from Guardian OS and Tandem Non-Stop UX OS to VAX/VMS and Windows NT [Gra90, Kal98, LI95, SK+00, SK+02, TI92, TI+95]. In server environments, Tandem computers, VAX clusters as well as several operating systems and file servers have been examined for software defects by several researchers. Lee and Iyer focussed on software faults in the Tandem GUARDIAN operating system [LI95], Tang and Iyer considered two VAX clusters running the VAX/VMS operating system [TI92], and Sullivan and Chillarege examined software defects in MVS, DB2, and IMS [SC91]. Murphy and Gent also focussed on system crashes in VAX systems over an extended period, almost a decade [MG95]. They concluded that system management was responsible for over 50% of failures with software trailing at 20% followed by hardware that is responsible for about 10% of failures.

While examining NFS data availability in Network Appliance's NetApp filers, Lancaster and Rowe attributed power failures and software failures as the largest contributors to downtime; operator failure contributions were negligible [LR01]. Thakur and Iyer examined failures in a network of 69 SunOS workstations [TI96]. They divided problem root causes into network, non-disk and disk-related machine problems. Kalyanakrishnam, et al. perused six months of event logs from a LAN comprising of Windows NT workstations that delivered emails [KK+99]. Using a state machine model of detailed system failure states to describe failure timelines on a single node, they concluded that most automatic system reboot problems are software-related; the average downtime is two hours. Similarly, Xu, et al. considered Windows NT event log entries related to system reboots for a network of workstations that were used for enterprise infrastructure, allowing operators to annotate event logs to indicate the reason for reboot [XK+99].

In this progression, our study of Windows' crash data gauges the evolution of PC reliability. Koopman, et al. test operating systems against the POSIX specification [KD00]. Our study is complementary to this work as we consider actual crash data that leads to OS unreliability.

Recently, in Windows XP Machines, Murphy deduced that display drivers were a dominant crash cause and memory is the most frequently failing hardware component [Mur04]. We extend this work by studying actual crash instances experienced by users rather than injecting artificial faults as performed by fuzz testing [FM00]. Our study of crash data differs from error log analysis performed by Kalakech, et al. [KK+04]; we determine the cause of crashes in addition to time and frequency.

Several researchers have provided insights on benchmarking and failure data analysis [BC+02, BS97, OB+02, WM+02]. Wilson, et al. suggest evaluating the relationship between failures and service availability [WM+02]. Among other metrics, when evaluating dependability, system stability is a key concern. Ganapathi, et al. examine Windows XP registry problems and their effect on system stability [GW+04]. Levendel suggests using the catastrophic nature of failures to evaluate system stability [Lev89]. Brown, et al. provide a practical perspective on system dependability by incorporating users' experience in benchmarks [BC+02, BS97]. In our study of crashes, we consider these factors when evaluating various applications.

## Overview of Crashes and Crashdumps

A crash is an event caused by a problem in the operating system (OS) or application (app) requiring OS or app restart. App crashes occur at user level and typically involve restarting the crashing application. An OS crash occurs at kernel-level, and is usually caused by memory corruption, bad drivers or faulty system-level routines. OS crashes are more frustrating than application crashes as they require the user to kill and restart the Windows Explorer process at a minimum, more commonly forcing a full machine reboot. While there are a handful of crashes due to memory corruption and other common systems problems, a majority of these OS crashes are caused by device drivers. These drivers are related to various components such as display monitors, network and video cards.

Upon each OS crash or bluescreen generated by the operating system, Windows XP collects failure data as a minidump. Users have three different options for the amount of information that is collected upon a crash. We use the default (and smallest) option of collecting small dumps, which are only 64K in size. These small minidumps contain a partial snapshot of the computer's state at the time of crash. They include a list of loaded drivers, the names and timestamps of binaries that were loaded in the computer's memory at the time of crash, the processor context for the stopped process, and process information and kernel context for the stopped process and thread as well as a brief stack trace. We do not collect personal data files for our study. However, portions of such data may be resident in memory at the time of crash and will consequently appear in our crash dumps. To disable personal data inadvertently being sent, crash reporting may be disabled or the user can choose not to send a particular crash report.

When an OS crash occurs, typically the entire machine must be rebooted. Any relevant information that can be captured before the reboot is saved in a .dmp file in the %windir%\Minidump directory. These minidumps are uniquely named with the date of the crash and a serial number to eliminate conflicting names for multiple crashes on the same day.

## Overview of BOINC Crash Collector

Berkeley Open Infrastructure for Network Computing (BOINC) is a platform for pooling computer resources from volunteers to collect data and run distributed computations [And03]. A popular example of an application using this platform is SETI@home, which aggregates computing power to 'search for extraterrestrial intelligence.' BOINC provides services to send and receive data from its users via the HTTP protocol using XML formatted files. It allows application writers to run and maintain a server that can communicate with numerous client machines through a specified Applications Programmer Interface (API). Each subscribed user's machine, when idle, is used to run BOINC applications. Project groups can create project web sites with registration services for users to subscribe and facilitate a project. The web site can also display statistics for contributing users.

Taking advantage of these efforts, we have created a data collection application to run on this platform. BOINC provides a good opportunity to collect and aggregate data from users outside our department while addressing privacy concerns. BOINC anonymizes user information while allowing us to correlate data from the same user. We have written tools to read minidumps from users' machines and send the data to our BOINC server. The drawback of this mechanism is that we can only collect crash dumps that are stored in known locations on the user's computer, consequently excluding application crash dumps that are stored in unknown app-specific locations. Furthermore, configuring the BOINC server is a tedious and meticulous task. We must also monitor the number of work units we allot for the BOINC projects; if there are not enough work units, the application will not run on client machines.

An attractive aspect of using BOINC is that we can add more features to our application as and when necessary. We can also provide users with personalized feedback pages, consequently rewarding the users with an incentive for sharing data. However, we must verify the integrity of each crashdump we receive from the users; users often create files in the crashdump directory to inflate their crash contribution ranking.

We use a combination of Microsoft's analysis tools and custom-written scripts to parse, filter and analyze the crash data. Received crash dumps are parsed using Microsoft's "Debugging Tools for Windows" (WinDbg), publicly available at http://www.microsoft.com/whdc/devtools/debugging/default.mspx. We retrieve debugging symbols from Microsoft's publicly available symbol server (http://www.microsoft.com/whdc/devtools/debugging/symbolpkg.mspx). Parsing crash dumps using WinDbg reveals the module that caused the crash as well as the proximate cause of the crash via an error code of the crashing routine. The drawback of this approach is that we rely on the completeness and accuracy of Microsoft's symbols. For legal reasons, Microsoft does not make third party debugging symbols available, especially those related to antivirus and firewall software.

We have conducted experiments and noted that 10% of crashdumps parsed with publicly available debugging symbols have different analysis results as compared to results when parsed with Microsoft's internal symbols. Microsoft-written components such as ntoskrnl take the blame for several third party and antivirus/firewall-related crashes.

Once crash dumps are parsed by WinDbg, the importance of filtering data is evident. When a computer crashes, the application or entire machine is rendered unstable for some time during which a subsequent crash is likely to occur. Specifically, if a particular piece of hardware is broken, or part of memory is corrupt, repeated use is likely to reproduce the error. It is inaccurate to double-count subsequent crashes that occur within the same instability window. To avoid clustering unrelated events while capturing all related crash events, we cluster individual crash events from the same machine based on temporal proximity of the events. The data that is collected can be used to gather a variety of statistics. We can provide insight to the IT team about the dominant cause of crashes in the organization and how to increase product reliability. We can also use crash behavior to track any potential vulnerability as frequent crashes may be a result of malware on the machine. In the long run, we may be able to develop a list of safe and unsafe hardware and software configurations and installation combinations that result in crashes.

## Understanding Crash Data

To study a broad population of Windows users, we studied data from public-resource computing volunteers. Numerous people enthusiastically contribute data to projects on BOINC rather than corporations as they favor a research cause. Additionally, users appreciate incentive either through statistics that compares their machine to an average BOINC user's machine, or through recognition as pioneering contributors to the project.

Currently, we have about 3500 BOINC users signed up to our project. Over the last year, we have received 2528 OS crashes from 617 of these users; several users experienced (and reported) multiple OS crashes while a majority of them reported zero or one crash. Users reporting no crashes most likely do not actively run the BOINC client on their machine.

According to results shown in Figure 1 most users experienced (submitted) only one crash; however, several users suffered multiple OS crashes. One user appears to have experienced over 200 OS crashes over the last year! The number is staggering considering that this data is for kernel-level crashes. Perhaps the user's user-mode crash counts are as bad, if not worse, considering there is more opportunity for variability in user-mode components.

First we analyze each crash as a unique entity to determine statistics on what components cause the Windows OS to crash often. Then, to understand how crashes on the same machine relate to each other, we carefully examined machines that experienced more than 5 kernel crashes within a 24 hour time period. In several cases, we observed the same crash occurring repeatedly (i.e., same fault in same module). There were also scenarios with crashes in various components interleaved with one another. We examine user behavior, temporal patterns and device driver software reliability to understand these crashes.

## A Human Perspective

The human user plays a huge role in the wear and tear of a computer. User-interaction is among the most difficult patterns to quantify. We extracted three distinct user-scenarios from examining crash sequences from our data:

- Case 1: The user retries the same action repeatedly, and consequently experiences the same crash multiple times. He believes the repetition will eventually resolve the problem (which may be true over a long period of time). In this scenario, the user's model of how things work is incomplete. He does not understand the complex dependencies within the system.
- Case 2: There is some underlying problem at a lower level that is causing various different crashes. For example, if the user has hardware problems, he is likely to have many more crashes in random components. In this case, the user is simply flustered with all the crashes and fixing each driver involved in each crash still will not resolve his problem; he will have to fix the root cause.
- Case 3: The user knows what the problem is and simply does not see an incentive to fixing it. For example, he might be using an old version of a driver for which an update is available. There are three conceivable explanations for not updating the crashing driver: a) fear of breaking other working components, b) laziness, and c) fear of getting caught with an illegal copy of software.

## A Temporal Perspective

There are factors beyond end user behavior that demonstrate inter-crash relationships. Figure 2 shows a distribution of the uptime between a machine reboot and a crash event. We observe that 25% of crashes occur within 30 minutes of rebooting a machine. 75% of crashes occur within a day of rebooting a machine. Perhaps shorter system uptime intervals indicate the trend of several consecutive related crashes.

Upon analyzing crash sequences on various machines, we observed various distinct temporal indicators of crash cause:

- <5 minute uptime: A crash that occurs within 5 minutes of rebooting a computer is most indicative of a boot-time crash. The crash is not likely to have been caused by a user action. These crashes are the most frustrating as there is very little the user can do between the time of reboot and the time of crash. The user may gain insight on such crashes by examining the boot log.
- 5 minutes-1 hour uptime: These crashes are more likely to be caused by a specific sequence of events initiated by the user (e.g., accessing a particular file from a corrupt disk segment).

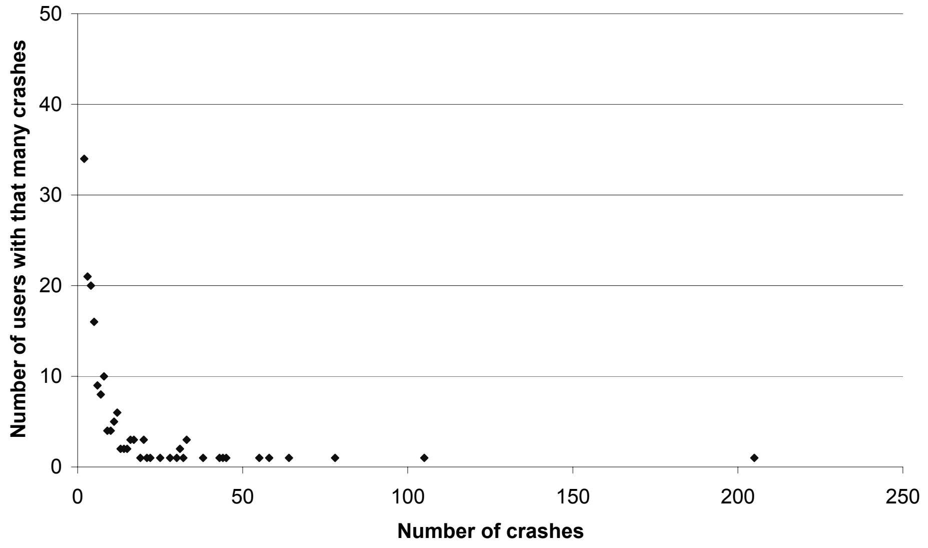


**Figure 1**: A histogram of the number of crashes experienced by users over the last year. One data point was omitted from the graph for clarity (443 users experienced only 1 crash each).

They could be attributed to software problems, hardware problems or memory corruption.

- Regular interval between crashes: Several users experienced crashes regularly at a particular time of day. Such crashes may be attributed to a periodic process resembling a cron job or an antivirus scan.
- Context-based: Various crashes are triggered by a logically preceding event. For example, every time a virus scanner runs, we may observe a failed disk access. In such scenarios, we cannot use exact time as an indicator.
- Random: Many crash sequences on users' machines did not fit in any of the above profiles. Several consecutive seemingly unrelated crashes could suggest a hardware problem and/or memory corruption.

Temporal crash patterns are useful in narrowing down a machine's potential root cause problems. However, the underlying responsibility of causing the crash lies in the longevity and reliability of the hardware and software on the machine.

## A Device Driver Reliability Perspective

Device drivers are a major contributor of kernel-level crashes. A device driver is a kernel-mode module that communicates operating system requests to the device and vice versa. These drivers are inherently complex in nature and consequently difficult to write. Among many reasons for device driver complexity are that these drivers deal with asynchronous events. Since they interact heavily with the operating system, the code must follow kernel programming etiquette (which is difficult to master and follow). Furthermore, once device drivers are written, they are exceedingly difficult to debug as the typical device driver failure is a combination of an OS event and a device problem, and thus very difficult to reproduce (see [SM+04] for a detailed description of device driver problems).

Figure 3 is largely based on the OS Crash Type field in analyzed crash reports. This field reveals graphics driver faults, common system faults (such as memory/pool corruption and hardware faults) and Application faults. However, there were many instances where the OS Crash Type was not provided (or defaulted to "Driver Fault") for legal reasons. In the absence of details revealed by the analysis tools, we crawled the web to derive the type of each driver that caused a crash. Where we were unable to determine the driver type (for example, when the documentation was not in English), we defaulted to "unknown."

Figure 4 shows that a handful of organizations contribute a significant number of crash-causing drivers to our data. Drivers written by seven organizations (Microsoft, Intel, ATI Technologies, Nvidia, Symantec, Zone Labs and McAfee) contributed 75% of all crashes in our data set. This trend suggests that crashes caused by poorly-written and/or commonly used drivers can be reduced significantly by approaching these top seven companies. On the other hand, the graph has a heavy tail, indicating that it would be extremely difficult to eliminate the remaining 25% of crashes as they are caused by drivers written by several different organizations.

Subsequently, we study the image (i.e., .exe, .SYS, or .dll file) that caused these crashes and identify the organization that contributed the crash-causing code see Figure 5.

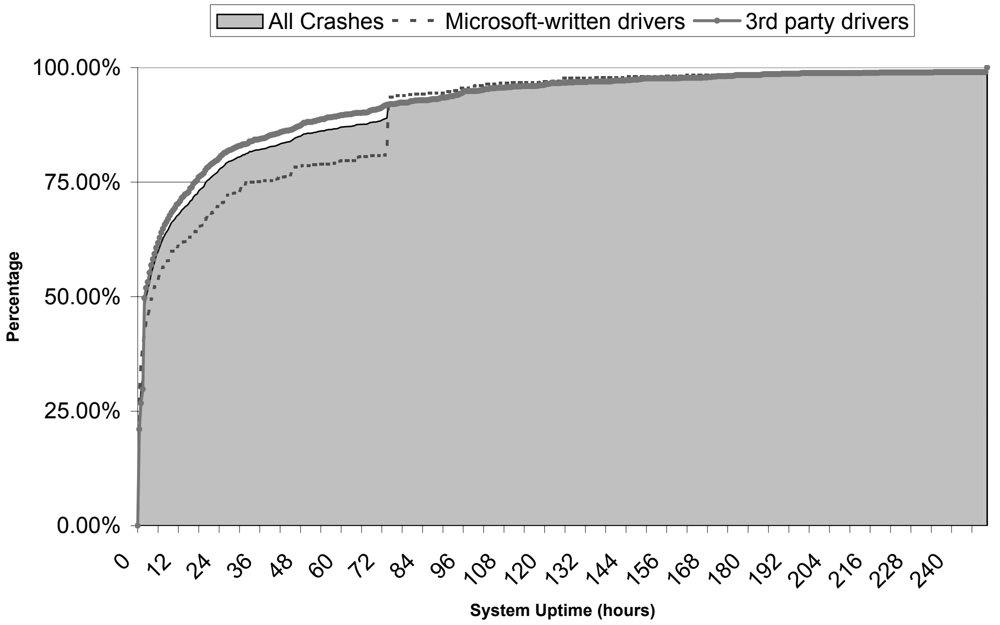


**Figure 2**: A cumulative frequency graph of system uptime between reboot and crash events. The dotted line extrapolates what the CFG would look like if Microsoft wrote all the drivers while the dashed line suggests what the CFG would look like if Microsoft wrote none of the drivers that crashed.

The top contender in Figure 5 is ialmdev5.dll, the Intel graphics driver. Recently, graphics drivers have become notorious for causing crashes and ialmdev5.dll is perhaps one of the more commonly used drivers in this category due to the popularity of Intel processors.

The second highest contender in Figure 5 is ntoskrnl.exe, which constitutes the bare-bones Windows NT operating system kernel code. It is not surprising that this executable is responsible for a number of driver crashes because it interacts with every other operating system component and is thus the single most critical component that can never be perfect enough. Furthermore, other systems code might generate bad input parameters to the ntoskrnl functions that cause exceptions; ntoskrnl bears the blame for the resulting crash as it generated the exception. Also, as mentioned earlier, many antivirus/firewall-related crashes may have been mis-categorized, blaming ntoskrnl due to third party privacy concerns (hence the significantly high percentage of crashes attributed to Microsoft in Figure 3).

Other crash causing images range from I/O drivers to multimedia drivers. It is difficult to debug or even analyze these crashes further as we do not have the code and/or symbols for these drivers.

| OS CRASH TYPE | NUMBER OF CRASHES |
|---|---|
| **OS Core** | **726** |
| Microsoft | 488 |
| Unknown | 238 |
| **Graphics Drivers** | **495** |
| Intel | 287 |
| ATI Technologies | 97 |
| Nvidia | 67 |
| Other | 44 |
| **Application Drivers** | **482** |
| Intel | 89 |
| Microsoft | 64 |
| Symantec | 58 |
| McAfee | 55 |
| Zone Labs | 55 |
| Unknown | 13 |
| Other | 148 |

| OS CRASH TYPE | NUMBER OF CRASHES |
|---|---|
| **Networking** | **338** |
| Unknown | 194 |
| Microsoft | 51 |
| Conexant | 17 |
| Other | 76 |
| **Common System Fault** (Hardware and Software Memory Corruption) | **136** |
| **Audio** | **130** |
| Avance Logic | 44 |
| C-Media | 33 |
| Microsoft | 16 |
| Other | 37 |
| **Storage** | **106** |
| Microsoft | 82 |
| Other | 24 |
| **Other** | **95** |
| **Unknown** | **20** |

**Figure 3**: Number of OS crashes of each type based on 2528 crashes received from BOINC users. (We would need many more samples before it would be safe generalizing these results to a larger user community.) This table also shows the top few crash-causing driver writers in each category.

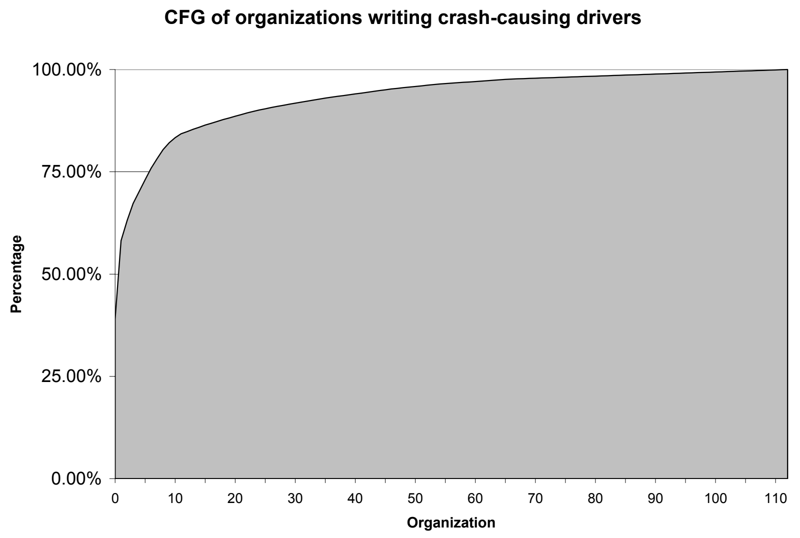


**Figure 4**: Cumulative Frequency Graph of organizations responsible for crash-causing drivers in our data. This graph does not account for driver popularity. 113 companies are represented in this graph.

With the increasing need for numerous devices accompanying the PC, it does not scale for the operating system developers to account for and write device driver code for each device; consequently, device drivers are written by device manufacturers, who are potentially inexperienced in kernel programming. Perhaps such lack of expertise is the most impacting cause for driver-related OS crashes.

We also observed numerous OS crashes caused by memory corruption. Memory corruption-related crashes can often be attributed to hardware problems introduced by the type of memory used (e.g., non-ECC memory). In the event that the memory corruption was due to software, the problem cannot be tracked down to a single image.

To further understand driver crashes, we studied the type of fault that resulted in the crash. Figure 6 lists the number of crashes that were caused by the various fault types. These fault types are reported by Microsoft's analysis tools when analyzing each OS crash dump.

While many of these fault types are straightforward to understand from the name, many others are abbreviations of the event they describe. Below, we enumerate each fault type and its significance (based on the descriptions provided in the parsed crash dumps):

- IRQL NOT LESS OR EQUAL – An attempt was made to access a pageable (or completely invalid) address at an interrupt request level (IRQL) that is too high. The driver is most likely using an improper address.[1]
- THREAD STUCK IN DEVICE DRIVER – The device driver is spinning in an infinite loop, most likely waiting for hardware to become idle. This usually indicates problem with the hardware itself or with the device driver programming the hardware incorrectly.

[1]The interrupt request level is the hardware priority level at which a given kernel-mode routine runs, masking off interrupts with an equivalent or lower IRQL on the processor. A routine can be preempted by an interrupt with a higher IRQL.

| Image Name/ Crash Cause | Image Description | Num Crashes | % Crashes | % Running Total |
|---|---|---|---|---|
| Ialmdev5.DLL | Intel graphics driver | 275 | 11% | 11% |
| ntoskrnl.exe | NT kernel and system | 187 | 8% | 19% |
| CAPI20.SYS | ISDN modem driver | 182 | 7% | 26% |
| Win32k.sys | multi user win32 driver | 114 | 5% | 31% |
| IdeChnDr.sys | Intel Application Accelerator driver | 89 | 4% | 35% |
| ntkrnlmp.exe | Multi-processor version of NT kernel and system | 87 | 4% | 39% |
| vsdatant.sys | TrueVector Device Driver | 51 | 2% | 41% |
| GDFSHK.SYS | McAfee Privacy Service File Guardian | 48 | 2% | 43% |
| V7.SYS | IBM V7 Driver for Windows NT/2000 | 45 | 2% | 45% |
| ALCXWDM.SYS | Windows WDM driver for Realtek AC'97 | 44 | 2% | 47% |

**Figure 5**: Top 10 OS Crash-causing Images based on 2528 crashes received from BOINC users. (We would need many more samples before it would be safe generalizing these results to a larger user community.) A description of the crash-causing image is provided in addition to the percentage of crashes caused by each image.

| Driver Fault Type | Num Crashes |
|---|---|
| IRQL NOT LESS OR EQUAL | 657 |
| THREAD STUCK IN DEVICE DRIVER | 327 |
| PAGE FAULT IN NONPAGED AREA | 323 |
| KERNEL MODE EXCEPTION NOT HANDLED | 305 |
| UNEXPECTED KERNEL MODE TRAP | 78 |
| BAD POOL CALLER | 74 |
| SYSTEM THREAD EXCEPTION NOT HANDLED | 73 |
| PFN LIST CORRUPT | 53 |
| DRIVER CORRUPTED EXPOOL | 38 |
| MACHINE CHECK EXCEPTION | 37 |

**Figure 6**: Top 10 crash generating driver fault types.

- PAGE FAULT IN NONPAGED AREA – Invalid system memory was referenced, for example, due to a bad pointer.
- KERNEL MODE EXCEPTION NOT HANDLED – The exception address pinpoints the driver/function that caused the problem. However, the particular exception thrown by the driver/function was not handled.
- UNEXPECTED KERNEL MODE TRAP – A trap occurred in kernel mode, either because the kernel is not allowed to have/catch (bound trap) the trap or because a double fault occurred.
- BAD POOL CALLER – The current thread is making a bad pool request. Typically this is at a bad IRQL level or double freeing the same allocation, etc.
- SYSTEM THREAD EXCEPTION NOT HANDLED – This fault type is similar to an unhandled kernel mode exception.
- PFN LIST CORRUPT – Typically caused by drivers passing bad memory descriptor lists.
- DRIVER CORRUPTED EXPOOL – An attempt was made to access a pageable (or completely invalid) address at an interrupt request level (IRQL) that is too high. This fault is caused by drivers that have corrupted the system pool.
- MACHINE CHECK EXCEPTION – A fatal machine-check exception occurred (due to hardware).

Studying these fault types reveals various programming errors that impact system behavior and what OS problems to tackle with caution. However, this information is more useful to the software developer than the end user. From a user's perspective, the most useful piece of information is "what can I fix on my machine?"

There are three distinct trends we observed on machines with multiple crashes:

- The same driver causes most crashes: This scenario is very simple to resolve. Most likely, the crash-causing driver is an old version, which has newer, more stable version available. There were other cases where a newly downloaded driver caused various crashes as a result of its incompatibility with other components installed on the machine. In both situations, updating or rolling back the driver's version will reduce crashes on the machine.
- Related drivers cause most crashes: Two drivers are considered related if they communicate with the same device or pertain to the same component. In this scenario if different, yet related, drivers cause the machine's crashes, then perhaps the common underlying component or device is at fault and needs attention.
- Unrelated drivers cause the crashes: This scenario is the most difficult to comprehend. First, we understand what the drivers have in common – whether they perform similar actions or function calls, have similar resource requirements (e.g., requiring network connectivity), or access the same objects.

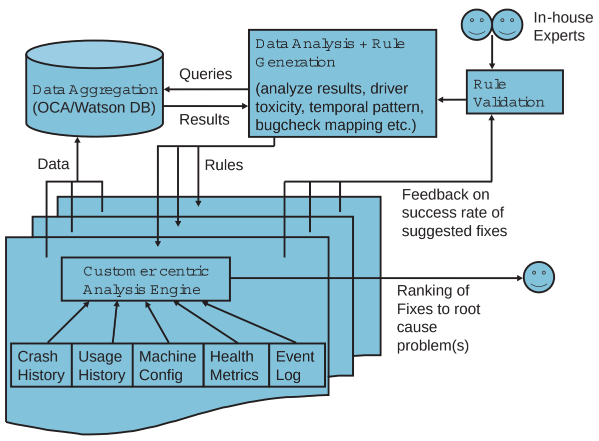
In the above scenarios, it is useful to understand inter-driver dependencies. We would also benefit from understanding the stability of specific driver versions and how diverse their install base is.

## Discussion

Windows users have started viewing crashes as a fact of life rather than a problem. We have the single most valuable resource to design a system that helps users cope with crashes better – crash data. Microsoft's Online Crash Analysis provides users with feedback on each of their submitted crashes. However, many users suffer from multiple crashes and individual per-crash analysis is not enough to identify the optimal solution to the root problem. There is a strong need to use historical data for each machine and use heuristics to determine the best fix for that machine.

The human, temporal and device-driver reliability perspectives shed light on potential root causes for crashing behavior. There are numerous other factors we can include to refine root cause analysis. It would be very beneficial to scrape portions of the machine's event log when analyzing crashes. We can look for significant events preceding each crash (e.g., Driver installed/removed, process started up, etc.), pinpointing likely sources of the machine's behavior.

It is also useful to collect various machine health metrics such as frequency of prophylactic reboot and frequency of virus scans. Such metrics will help us evaluate the relative healthiness of a machine (compared to the entire user population) and customize analysis responses on a per-machine basis. Ideally we would want our data analysis system to have a built-in feedback loop (as seen in Figure 7) so we can continuously adapt and improve our analysis engine. This framework is useful for performing accurate post-mortem analysis.



In-house Experts
Data Analysis + Rule Generation
(analyze results, driver toxicity, temporal pattern, bugcheck mapping etc.)
Queries
Results
Data Aggregation (OCA/Watson DB)
Rule Validation
Data
Rules
Feedback on success rate of suggested fixes
Customer centric Analysis Engine
Ranking of Fixes to root cause problem(s)
Crash History
Usage History
Machine Config
Health Metrics
Event Log

**Figure 7**: Customer-centric kernel crash analysis framework.

It is equally important to understand the manifestation of such problems on each machine. It is important to characterize inter-component interactions and model failure propagation patterns. Such analysis will help improve inter-component isolation, reducing their crash-likeliness. While post-mortem analysis and debugging helps cure problems, it is also critical to prevent problems at their source. As an industry, we must work towards determining the characteristics of software that dictate software dependability.

## Conclusion

Our crash-data related study, despite the small quantity of Windows XP data analyzed, has contributed several observations. The most notable reality is that the Windows operating system is not responsible for a majority of PC crashes in our data set. Poorly-written device drivers contribute most of the crashes in our data. It is evident that targeting a few companies to improve their driver quality will effectively eliminate 75% of our crashes. However, the remaining 25% of crashes are extremely difficult to eliminate due to the large number of organizations contributing the driver code.

Users can alleviate computer frustration by better usage discipline and avoiding unsafe applications and drivers. With additional data collection and mining, we hope to make stronger claims about applications and also extract safe product design and usage methodology that apply universally to all operating systems. Eventually, this research can gauge product as well as usage evolution.

Studying failure data is as important to the computing industry as it is to consumers. Product dependability evaluations help evolve the industry by reducing quality differential between various products. Once product reliability data is publicized, users will use such information to guide their purchasing decisions and usage patterns. Product developers will react defensively and resulting competition will improve quality control.

In the future, we hope to refine analysis engine and automate many of the background queries for each driver. We would like to improve our understanding of the dependencies between analysis categories such as the temporal and device driver perspectives. We also plan to investigate the relationship of various objects involved at the time of crash. Lastly, we would like to obtain more environmental metrics and draft more rules for analysis, and extend this work to other Operating Systems.

## Author Biographies

Archana Ganapathi is a graduate student at the University of California at Berkeley. She completed her Masters degree in 2005 and is currently pursuing her Ph.D .in Computer Science. Her primary research interests include operating system dependability and system management.

Viji Ganapathi is an undergraduate at the University of California at Berkeley. She will complete her Computer Science Bachelors degree in December, 2006.

David A. Patterson has been Professor of Computer Science at the University of California, Berkeley since 1977, after receiving his A.B., M.S., and Ph.D. from UCLA. He is one of the pioneers of both RISC and RAID, both of which are widely used. He co-authored five books, including two on computer architecture with John L. Hennessy. They have been popular in graduate and undergraduate courses since 1990. Past chair of the Computer Science Department at U.C. Berkeley and the Computing Research Association, he was elected President of the Association for Computing Machinery (ACM) for 2004 to 2006 and served on the Information Technology Advisory Committee for the U.S. President (PITAC) from 2003 to 2005.

His work was recognized by education and research awards from ACM and IEEE and by election to the National Academy of Engineering. In 2005 he shared Japan's Computer & Communication award with Hennessy and was named to the Silicon Valley Engineering Hall of Fame. In 2006 he was elected to the American Academy of Arts and Sciences and the National Academy of Sciences and he received the Distinguished Service Award from the Computing Research Association.

## Bibliography

[And03] Anderson, D., "Public Computing: Reconnecting People to Science," *The Conference on Shared Knowledge and the Web*, Residencia de Estudiantes, Madrid, Spain, Nov., 2003.

[BS+02] Broadwell, P., N. Sastry and J. Traupman, "FIG: A Prototype Tool for Online Verification of Recovery Mechanisms," *Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN)*, New York, NY, June, 2002.

[BC+02] Brown, A., L. Chung, and D. Patterson, "Including the Human Factor in Dependability Benchmarks," *Proc. 2002 DSN Workshop on Dependability Benchmarking*, Washington, D.C., June, 2002.

[BS97] Brown, A. and M. Seltzer, "Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture," *Proc. 1997 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Seattle, WA, June, 1997.

[FM00] Forrester, J. and B. Miller, "An Empirical Study of the Robustness of Windows NT Applications Using Random Testing," Proc. 4th USENIX Windows System Symposium, Seattle, WA, Aug., 2000.

[G05] Ganapathi, A. "Why Does Windows Crash?" *UC Berkeley Technical Report UCB//CSD-05-1393*, May, 2005.

[GW+04] Ganapathi, A., Y. Wang, N. Lao and J. Wen, "Why PCs are Fragile and What We Can Do About It: A Study of Windows Registry Problems," *Proc. International Conference on Dependable Systems and Networks (DSN-2004)*, Florence, Italy, June, 2004.

[GP05] Ganapathi, A. and D. Patterson, "Crash Data Collection: A Windows Case Study," To Appear in *Proc. International Conference on Dependable Systems and Networks (DSN-2005)*, Yokohama, Japan, June, 2005.

[Gra86] Gray, J., "Why Do Computers Stop and What Can Be Done About It?" *Symp. on Reliability in Distributed Software and Database Systems*, pp. 3-12, 1986.

[Gra90] Gray, J., "A census of Tandem system availability between 1985 and 1990," *Tandem Computers Technical Report 90.1*, 1990.

[GS04] Gray, J. and A. Szalay, "Where the rubber meets the sky:bridging the gap between databases and science," *Microsoft Research TR-2004-110*, 2004.

[KK+04] Kalakech, A., K. Kanoun, Y. Crouzet and J. Arlat, "Benchmarking the dependability of Windows NT4, 2000 and XP," *Proc. International Conference on Dependable Systems and Networks (DSN-2004)*, Florence, Italy, June, 2004.

[Kal98] Kalyanakrishnam, M., "Analysis of Failures in Windows NT Systems," Masters Thesis, *Technical report CRHC 98-08*, University of Illinois at Urbana-Champaign, 1998.

[KK+99] Kalyanakrishnam, M., Z. Kalbarczyk, and R. Iyer, "Failure data analysis of a LAN of Windows NT based computers," *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, 1999.

[KD+05] King, S., G. Dunlap and P. Chen, "Debugging operating systems with time-traveling virtual machines," *Proceedings of the 2005 Annual USENIX Technical Conference*, April, 2005.

[KD00] Koopman, P., and J. DeVale, "The Exception Handling Effectiveness of POSIX Operating Systems," *IEEE Transactions on Software Engineering*, Vol. 26, Num. 9, pp. 837-848, Sept., 2000.

[LR01] Lancaster, L. and A. Rowe, "Measuring real-world data availability," *Proceedings of LISA 2001*, 2001.

[LI95] Lee, I. and R. Iyer, "Software Dependability in the Tandem GUARDIAN Operating System," *IEEE Transactions on Software Engineering*, Vol. 21, Num. 5, pp. 455-467, May, 1995.

[Lev89] Levendel, Y., "Defects and Reliability Analysis of Large Software Systems: Field Experience," *Digest 19th Fault-Tolerant Computing Symposium*, pp. 238-243, June, 1989.

[MR+04] Maniatis, P., M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, and M. Baker, "The lockss peer-to-peer digital preservation system," *ACM Transactions on Computer Systems (TOCS)*, 2004.

[Mur04] Murphy, B., "Automating Software Failure Reporting," *ACM Queue*, Vol. 2, Num. 8, Nov., 2004.

[MG95] Murphy, B. and T. Gent, "Measuring system and software reliability using an automated data collection process," *Quality and Reliability Engineering International*, Vol. 11, 1995.

[OB+02] Oppenheimer, D., A. Brown, J. Traupman, P. Broadwell, and D. Patterson, "Practical issues in dependability benchmarking," *Workshop on Evaluating and Architecting System dependabilitY (EASY '02)*, San Jose, CA, Oct., 2002.

[SS72] Schroeder, M. and J. Saltzer, "A Hardware Architecture for Implementing Protection Rings," *Communications of the ACM*, Vol. 15, Num. 3, pp. 157-170, March, 1972.

[SK+00] Shelton, C., P. Koopman, K. DeVale, "Robustness Testing of the Microsoft Win32 API," *Proc. International Conference on Dependable Systems and Networks (DSN-2000)*, New York, June, 2000.

[SK+02] Simache, C., M. Kaaniche, A. Saidane, "Event log based dependability analysis of Windows NT and 2K systems," *Proc. 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02)*, pp. 311-315, Tsukuba, Japan, Dec., 2002.

[SC91] Sullivan, M. and R. Chillarege, "Software defects and their impact on system availability – a study of field failures in operating systems," *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, 1991.

[SM+04] Swift, M., Muthukaruppan, B. Bershad, and H. Levy, "Recovering Device Drivers," *Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec., 2004.

[TI92] Tang, D. and R. Iyer, "Analysis of the VAX/VMS Error Logs in Multicomputer Environments – A Case Study of Software Dependability," *International Symposium on Software Reliability Engineering*, Research Triangle Park, North Carolina, Oct., 1992.

[TI96] Thakur, A. and R. Iyer, "Analyze-NOW-an environment for collection and analysis of failures in a network of workstations," IEEE Transactions on Reliability, Vol. R46, Num. 4, 1996.

[TI+95] Thakur, A., R. Iyer, L. Young and I. Lee, "Analysis of Failures in the Tandem NonStop-UX Operating System," *International Symposium on Software Reliability Engineering*, Oct., 1995.

[WL+93] Wahbe, R., S. Lucco, T. Anderson, and S. Graham, "Efficient Software-Based Fault Isolation," *Proc. Fourteenth ACM Symposium on*

*Operating Systems Principles (SOSP)*, pp. 203-216, December, 1993.

[WC+01] Welsh, M., D. Culler and E. Brewer, "SEDA, an Architecture for well-conditioned scalable Internet Services," *18th Symposium on Operating System Principles*, Chateau Lake Louise, Canada, October, 2001.

[WM+02] Wilson, D., B. Murphy and L. Spainhower, "Progress on Defining Standardized Classes for Comparing the Dependability of Computer Systems," *Proc. DSN 2002 Workshop on Dependability Benchmarking*, Washington, D.C., June, 2002.

[XK+99] Xu, J., Z. Kalbarczyk and R. Iyer, "Networked Windows NT system field failure data analysis," *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing*, 1999.