Using Computers to Diagnose Computer Problems

Joshua A. Redstone, Michael M. Swift, Brian N. Bershad
Department of Computer Science and Engineering
University of Washington, Seattle
(redstone, mikesw, bershad)@cs.washington.edu

Abstract

Although computers continue to improve in speed and functionality, they remain difficult to use. Problems frequently occur, and it is hard to find fixes or workarounds. This paper argues for the importance and feasibility of building a global-scale automated problem diagnosis system that captures the natural, although labor intensive, workflow of system diagnosis and repair. The system collects problem symptoms from users' desktops, rather than requiring users to describe their problems to primitive search engines, automatically searches global databases of problem symptoms and fixes, and also allows ordinary users to contribute accurate problem reports in a structured manner.

1 Introduction

Despite continuous advances in hardware and software technology, computers are still difficult to use. They often behave in unexpected ways, and it is hard to find fixes or workarounds for problems encountered. The typical approach to solving a problem is to describe the symptoms (e.g. "Word footnotes don't work") to the keyword search interface of a vendor-owned help database, a small number of public databases, and then finally a broad "Google"-like search of the entire web. With luck, the "right" choice of keywords may quickly produce an article or posting describing the problem, the cause, and hopefully a resolution. More likely, though, the user gets back too little, too much, or the wrong information. He may continue searching, contact customer support or a message board, or simply give up and hope the problem doesn't come up again. This can be time consuming and frustrating. Moreover, for a given problem, this diagnostic process is repeated for each user touched by the problem, leading to massive global costs as a single problem is diagnosed millions of times. In contrast, root cause analysis and repair is done relatively infrequently. Once a user or company has identified a problem's symptoms and repair procedures, he or it posts a solution to some database, with the intent of sharing the solution with everyone. Unfortunately, the high global cost of finding the solution substantially reduces the value of having it in the first place.

Today, computers scan genetic sequences to identify the root causes of disease, pinpoint DDOS attacks on the Internet, and even match up lonely singles based on personality profiles. Yet, they are nearly useless when our computers don't do what we expect, even when the same problem has occurred a thousand times before on a thousand different machines.

This paper presents the simple vision in which computers diagnose their own problems, leveraging prior analysis work done by others. In line with this vision, we propose that problem reports, which today are unstructured text, follow a structured format and in particular that they express symptoms and causes in a machine-readable and machine-testable format. A structured representation simplifies diagnosis since an errant client machine can search for and test itself against symptoms in a global database with high precision. The structured representation, does, however, complicate the task of problem reporting. While true, we believe that finding and fixing a problem for the first time is where the hard work occurs, and that any incremental burden posed by representing that process in a structured format is small.

1.1 Why now and not before?

In the history of computing, there's never been a time that the system, and not the user, has been responsible for closing the gap between system behavior and user expectations. Why, then, is *now* the right time to start building automated, global-scale, diagnostic services? Consider these enabling factors:

- First, the Internet generates the ultimate network effect, making it possible for anyone, anywhere on the planet, to derive value from the prior experiences (good and bad) of others. Proxy caches, peer-to-peer file sharing networks and even user-contributed product reviews have shown this.

- Second, although there are hundreds of millions of machines, there is relatively little variety in hardware and software. Consequently, a problem found and fixed on one machine is likely to be found and fixed in the same way on another.
- Third, many operating systems support a generalized form of configuration management, such as the Windows registry. This makes it feasible to automatically determine the configuration of a machine.
- Fourth, standardized user-interfaces facilitate the mechanical recording of user-interface events and hence discovery of symptoms.
- Fifth and finally, vendors are making more of their bug databases public (their paranoia is mitigated by the economic benefit of avoiding direct contact with the customer), so there exist large, high-quality sources of known problems that can be automatically diagnosed.

Because of these reasons, an effective diagnosis solution can be built today, using existing operating systems and applications.

However, as we will discuss, there is an opportunity to build an even more effective solution by extending today's operating systems and applications so that a diagnosis engine can observe all state and behavior of the computer, across all applications. At first glance, it may seem as though any additional operating system or application work to support automated diagnosis creates a new burden, and therefore development cost, to be borne by software manufacturers. However, we argue that automated diagnosis is actually a necessary component of any system claiming *high availability* as a feature. Although availability is typically measured in terms of uptime – how long since the last crash – this system-oriented perspective is irrelevant to the user. Instead, a user perceives availability in terms of *goodtime*, which is uptime less the amount of time spent figuring out why the system isn't doing what the user expects. As an example, one of the authors of this paper recently switched his day-to-day working platform from Windows XP to another operating system. Although Windows XP uptime was much improved over its predecessors, goodtime was not. In contrast, uptimes on this other operating system are roughly the same as with Windows XP, but the goodtimes are better. While we don't intend to justify our position with a single anecdote, it should be clear that the broad platform coverage of most applications is becoming such that users can easily migrate to those systems where goodtimes are plentiful.

The goal of this paper is to encourage the systems community to take seriously the challenge of automatically diagnosing system and application problems, and to show that there exists a reasonable path that gets us from here to there. To be clear, it is a path that we ourselves have not traveled as we have not built the system we describe. Consequently, our assumptions are unchecked, and there may be some very good reasons why the state of the art in systems diagnostics is and will remain a glorified version of *grep*. But, it seems unlikely.

In the next section, we present additional background material so that the reader can differentiate from what has been and what remains to be done. In Section 3 we sketch out one possible solution to the problem of automated diagnostics. Finally, in Section 4 we conclude.

2 Background

Software vendors, in order to reduce customer support costs, are, and have been, motivated to provide some sort of diagnostic function with their systems. Most primitively, one can even find occasional bug reports at the bottom of some thirty-year-old UNIX man pages. Where the software vendors have left off, user communities have picked up with their own FAQs and bulletin boards. Indeed, the theme of "why bad things happen to good computers" has spawned an entire book genre dedicated to more goodtime. Below, we briefly characterize a few existing solutions according to whether they are manual or automatic.

2.1 Manual diagnosis

Manual problem diagnosis has the user searching public information sources for a problem report. There are two common information sources: vendor-controlled databases (such as [1] or [11]) and community databases (such as [3] or [16]). Vender-controlled databases have the advantage of providing high-quality coverage of a specific class of problems. However, they are limited in scope and are closed – contribution is tightly controlled. Limiting contribution means that database information may be stale and may contain omissions due to broader corporate considerations.

Community databases, such as discussion boards and mailing lists, offer more wide-ranging and up-to-date information. However, with no standard format for articles, locating information is especially difficult. In addition, information may be inaccurate because there is no quality control mechanism. Posting to these forums can be effective, but often requires an extended dialog to describe key symptoms and configuration details.

Searching is difficult in both kinds of databases because the search interface is inefficient and error prone. Most systems offer only keyword search, although a few natural language systems exist [7]. Successful keyword search requires choosing the correct terminology, which in turn frequently requires

detailed technical understanding of the problem. Further, that terminology may not be the same for all databases. Natural language systems promise to improve search quality, but still require sufficiently detailed understanding of the problem to formulate a specific request.

Even when a user's search locates a possible cause report, the user must manually determine if the system diagnosed in the report is "phylogenetically similar" to the system in question. Often, this is impossible, as the information in the report insufficiently describes the elements of the reported system. At other times, the system may be well-described, leaving it to the user to determine the differences between the systems, and then if they matter (e.g., difference in BIOS versions). For example, a printer may not print because a user has the wrong driver for the printer, or because the driver is installed in the wrong directory, with each possible cause described in a separate report. A user who has the wrong driver installed in the wrong directory will unprofitably apply the fix from one report without considering the second.

2.2 Automated diagnosis

An automated diagnosis service shields the user from the details of determining the source of a problem, and focuses instead on revealing the solution. Broadly, there are two types of problems dealt with in automated systems:

Type I: These problems are those for which the resolution is to change the system (upgrading an application, fixing the registry, etc.), and typically result from some well, or partially, understood bug.

Type II: These problems are those for which the resolution is to change user behavior (e.g., saving as a different file format). These problems are either due to correct but undesired system behavior or to a bug for which a fix is not known.

Addressing Type I problems are systems such as WindowsUpdate [14], Windows Baseline Security Analyzer [10] and virus scanners [5], which scan a computer and list available software updates or fixes. However, these solutions lack the ability to diagnose specific problems (they find all bugs with known fixes rather than the bug you want fixed), and the first two may introduce new problems by fixing bugs that are not experienced. By analogy, one can imagine a general practitioner who prescribes a lung transplant in order to cure a patient's nagging cough.

The Windows Error Reporting System ([12] and [13]) suggests fixes for Type I problems, but only those that cause crashes, which occur much less often than general usability problems. Upon a crash, it alerts Microsoft, reporting the loaded executables and their versions. In some cases, Microsoft is able to identify the bug and provide a fix immediately. Autonomic Computing [2][9] also addresses only Type I problems by monitoring system behavior, and then tuning or repairing the system as appropriate. The operative analogy with these systems is the mechanic who replaces your car's brakes after you've run into a wall because you were distracted trying to figure out how to turn off the windshield wipers. The problem is with the wipers, not the brakes.

Agent based approaches such as the Microsoft Office Assistant [8] target the subset of Type 2 problems that do not include bugs. Agents passively monitor user activity and actively offer suggestions based on observed behavior. Such systems are a step in the right direction, but we believe that greater depth, coverage, and specificity are required.

To summarize, existing manual and automated tools provide only limited assistance in diagnosis. Manual tools can diagnosis a wide range of problems but imprecisely and at high cost. Automated tools provide low-cost assistance, but only cover a limited range of problems and do not provide targeted assistance for particular problems.

3 Automated Problem Diagnosis

An automatic problem diagnosis system has three components. The first component captures aspects of the computer's state and behavior necessary to characterize the problem. This includes the symptoms and information such as the installed applications and their versions. The second component matches this information against problem reports to identify the report(s) that best diagnose the problem. Finally, the third component produces new problem reports as new problems are discovered. This section discusses issues in the design of each component.

3.1 Observing Symptoms

The flexibility of a diagnosis system to handle different problems depends on its ability to observe the problem. This is a challenging task. To collect the information necessary to perform a diagnosis, all relevant information must be visible to the diagnosis engine. For example, diagnosing a bug in which a missing library causes an application to display an error message when launched involves observing the error message and that the library is missing. If the tool cannot observe the error message or detect the library's existence, it cannot diagnosis this particular problem. In addition, state and behavior should be observable at the correct level of abstraction for robust diagnosis. For example, observing that the 'OK' button was clicked may be more useful than observing that a mouse click occurred at pixel (x,y). Translation between abstraction levels is possible, but may be difficult to accomplish in a robust manner.

Computers today already expose a wide variety of state and behavioral information, allowing for the diagnosis of a wide range of problems. However, gaps exist, and ultimately, robust diagnosis will require changing systems and applications to reveal more information. The following paragraphs discuss the information available today. We divide the discussion of observable state and behavior information into three categories depending on the agent: the user, applications, or the operating system.

User behavior is revealed in terms of the user's input to the system. Since input is an OS service, it is readily available for observation. Most systems allow capture of mouse and keyboard events. Many applications use standard user-interface toolkits, which interact with system resources in an easily observable way. For these applications, events such as menu and dialog box activity are visible. However, observing user behavior in applications that manage their own user interface requires their cooperation in some form.

Applications serendipitously expose a fair amount of behavior and state by virtue of their interaction with the operating system (and thus, hardware). For example, it is straightforward to capture calls to system APIs, including registry (on Windows) and file system accesses. However, applications will not reveal much internal information without modification. Many applications support a debug interface, usually used during development, or export an API for extensibility. Exploiting this functionality may offer a low cost path to accessing more internal application state and behavior.

The operating system exposes many aspects of state and behavior already for the purpose of system management, through performance monitors and event logs. Furthermore, useful configuration information is available from the file system and registry (in Windows) or /etc (in *nix). In addition, the OS exports a rich programming interface for discovering and manipulating system state. Therefore, tools can adequately capture OS behavior and configuration today.

Diagnosis fundamentally involves gathering information, which raises privacy concerns. Users today explicitly choose which information to share with support forums or help desks, and the diagnosis system should provide similar options. If the diagnosis process involves an untrusted computer, then system state must be filtered to avoid disclosing sensitive data. Or, the diagnosis process can be performed only on trusted computers (e.g., by downloading problem databases.)

Examples

We briefly give three examples to show how an automated diagnosis system might detect that a user is having problems. The first example shows what can be done using today's system and application infrastructure. The second illustrates why changes to infrastructure may be required. Finally, the third problem is representative of a class of problems which we believe are not detectable using automatic means.

Problem 1: Quicktime is not installed. When a user clicks on a Quicktime URL, she expects a movie to play. Using today's system and application interfaces, a diagnostic service could observe the HTTP request for a URL ending in `.mov`, and a subsequent dialog box containing the message that there is no application to display this object. This is sufficient to scan the problem reports database and determine that Quicktime needs to be installed.

Problem 2: Page doesn't render properly. A more difficult problem might be that an HTML page won't render properly. Suppose that the particular cause is that too many display elements exceeds the browser's internal limits. For a diagnosis tool to recognize this, the browser would have to export sufficient information on its internal state and dynamic document contents.

Problem 3. Page doesn't print properly. Finally, diagnosing some problems, such as those that reveal themselves externally to the system, will be difficult without significant additional infrastructure. Consider a bug in which a document prints incorrectly. The information necessary to diagnose the problem, namely the printed document, is not visible to the computer. Even if the computer could observe the document, it may be challenging to characterize in a general way the qualities of the document that make it unsatisfactory. It may be necessary to involve the user here.

3.2 A Structured Database

The second element in a diagnosis system is a mechanism to search problem reports for those that match the observed state and behavior. We believe the way to accomplish this is to require that problem reports be written in a structured, machine-readable format, such as XML. XML provides a semi-structured format for reports with variable levels of detail, and a rich query language for matching symptoms to reports [4]. Future advances in natural language techniques may allow more flexible problem report formats. Automated capture of symptoms for both searching and reporting greatly increases the accuracy of searches compared to today's manual searches.

Searching the database, at the conceptual level, consists of comparing each problem report to the state and symptoms of the system. The reports can then be ranked according to how well they match.

3.3 Generating New Reports

The third and final element of the system generates new problem reports. While automated diagnosis is useful even if it can only recognize a few problems, the technique works best when leveraging the experiences of all who experience problems. The key is to make problem reporting as simple as possible. Manual entry of reports is always an option, but we believe that the technology of the other diagnosis components can aid this process. For example, when a user attempts to diagnose a problem and no problem report is found, the state and behavior information collected to aid in diagnosis can be used to construct a new problem report. There are similar privacy concerns when generating new reports as when collecting system state for diagnosis. The ultimate goal is to construct new reports without any user intervention at all. To distinguish the quality of reports, since they come from many sources, a community rating system, such as used at eBay.com [6], can be used.

3.4 Summary

Each of the three components of problem diagnosis described here has been inspired by counterpart work in other fields, which provides comfort that the ultimate solution is achievable. The TIGR [18] and NCBI [15] genome database projects aggregate gene data from many researchers, and leverage a standard data format to allow users to easily benefit from every researcher's contributions. The SDSS SkyView astronomy database [17] provides advanced online query access to astronomical data. The intelligent query interface provides efficient access to a large amount of data. eBay.com [6] has a feedback system to judge the quality of data from uncontrolled sources. Finally, the Windows Error Reporting Service [12] relies on technology to automatically construct new problem reports when users experience a crash without input from the user.

4 Conclusion

Despite continuous advances in hardware and software, computers are still difficult to use and users frequently have problems. Computer usability is essentially an availability issue: if the user can't get their job done, it is irrelevant that the computer is running. The correct metric of computer availability therefore is not the traditional *uptime* metric, but instead is the user's ability to get work done (*goodtime*).

This paper argues that an important way to increase *goodtime* is to decrease the time spent diagnosing problems by automating the diagnosis process. When the user encounters a problem, the computer should examine the state and behavior of the machine, search problem databases for a matching problem report, and present the diagnosis to the user. Ideally, this process should occur without any interaction from the user.

We believe that automating problem diagnosis is possible for a large class of problems with today's operating systems and applications, and can only improve with further operating system and application support. The challenges in building an automated diagnosis system are capturing relevant state and behavior, matching to problem reports, and creating new problem reports as new problems arise. Capturing state and behavior is possible because critical systems and applications already have interfaces and logs for determining the state and activities of a system. However, some problems are undetectable with the information available from applications and the operating system today. To perform robust detection, we need new interfaces in systems to allow applications to expose their internal state and behavior. Performing matching automatically is possible if we structure problem reports in a machine-readable format.

The state of system and application support for automated problem diagnosis today is not unlike the state of support for system auditing and event recording in early operating systems which were extraordinarily difficult to monitor. Then, to enable even the simplest of management functionality, applications and the operating had to change to expose their behavior to monitoring services via mechanisms such as SNMP. Similarly, automated problem diagnosis also requires application and OS modification. As with those earlier manageability overhauls, we believe that small investments in the way we build applications and operating systems will yield big returns.

5 References

[1] Apple Corporation. *AppleCare Knowledge Base*, http://kbase.info.apple.com.

[2] G. Banga. *Auto-diagnosis of Field Problems in an Appliance Operating System*, in Proceedings of the USENIX Technical Conference, June 2000.

[3] BugNet. *BugNet*, *http://www.bugnet.com*.

[4] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Sim6on and M. Stefanescu. *XQuery 1.0: An XML Query Language*, W3C Consortium, http://www.w3.org/TR/xquery.

[5] D. Chess. *Virus Verification and Removal Tools and Techniques*, Virus Bulletin, November 1991.

[6] EBay Corporation. *Feedback Profiles*, http://www.ebay.com.

[7] D. Heckerman and E. Horvitz. *Inferring Informational Goals from Free-Text Queries: A Bayesian Approach*, in Proceedings of the

Fourteenth Conference on Uncertainty in AI, July 1998.

[8] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. *The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users*, in Proceedings of the Fourteenth Conference on Uncertainty in AI, July 1998.

[9] IBM Corporation. *Autonomic Computing*, http://www.research.ibm.com/autonomic.

[10] Microsoft Corporation. *Microsoft Baseline Security Analyzer*, http://www.microsoft.com/technet/security/tools/Tools/MBSAhome.asp.

[11] Microsoft Corporation. *Microsoft Knowledge Base*, http://support.microsoft.com.

[12] Microsoft Corporation. *Windows Error Reporting*, http://msdn.microsoft.com/library/en-us/debug/base/windows_error_reporting.asp.

[13] Microsoft Corporation. *Windows Online Crash Analysis*, http://oca.microsoft.com/en/Welcome.asp.

[14] Microsoft Corporation. *Windows Update*, http://www.windowsupdate.com.

[15] J. Ostell and J. Kans. *The NCBI data model*. Methods of Biochemical Analysis, Vol. 39, July 1998.

[16] Redhat Corporation. *Redhat Support Forums*, http://www.redhat.com/support/forums.

[17] A. Szalay, J. Gray, A. Thakar, P. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg. *The SDSS SkyServer – Public Access to the Sloan Digital Sky Server Data*, in Proceedings of ACM SIGMOD, June 2002.

[18] TIGR. *The Institute for Genomic Research*. http://www.tigr.org.