Persistent-state Checkpoint Comparison for Troubleshooting Configuration Failures

Yi-Min Wang
Chad Verbowski
Daniel R. Simon

April 4, 2003

Technical Report
MSR-TR-2003-28

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

To appear in *Proc. IEEE International Conference on Dependable Systems and Networks (DSN)*, June 2003.

Persistent-state Checkpoint Comparison for Troubleshooting Configuration Failures

Yi-Min Wang Chad Verbowski Daniel R. Simon

Microsoft Research, Redmond, WA

Abstract

Application failures characterized by the phrases, "it worked yesterday, but it doesn't work today" and "it worked on that machine, but it doesn't work on this machine" are a major source of computer user frustration and a major component in the total cost of ownership. The typical symptom-based troubleshooting approach relies too much on creative thinking and may lead users or support technicians in directions far from the actual root cause. In this paper, we propose a state-based troubleshooting approach for configuration failures that aims at making the diagnostic process as mechanical as possible. In the narrow-down phase, we use checkpoint comparison and application tracing to determine which pieces of persistent state have changed and are affecting current application execution; ongoing self-monitoring of persistent-state changes by the machine is used to help eliminate false positives. In the solution-query phase, state-to-task mapping and searches of online databases are used to translate low-level state information into high-level user interfaces and articles. We describe the design and implementation of a troubleshooter that uses this state-based approach and present preliminary results to demonstrate its effectiveness in diagnosing several actual configuration failures.

1. Introduction

A program execution starts by creating a process from binary executables, reading configuration data from persistent storage, and accessing volatile resources (e.g., CPU, memory, etc.) through abstractions provided by the operating system. The execution can fail due to faults in any of these components. Failed program executions caused by Bohrbugs [GR93] (i.e., software faults that deterministically result in failures) can only be repaired through debugging and removal of the defects. Failed executions caused by Heisenbugs can often be repaired by restarting the program or rebooting the machine to provide a different volatile resource environment. In between is an interesting class of software failures caused by configuration problems: they fail deterministically across restarts, but they can be repaired by correcting the configuration problems, without modifying the program executables.

We use the following terminology [SS92] in this paper: a *configuration fault* refers to a piece of faulty configuration data; a *configuration error* is the manifestation of a configuration fault that causes the program state to deviate from the correct one; and the error may eventually result in a *configuration failure* when the incorrect state results in an externally observable symptom such as an error dialog box or failure to deliver the expected service. A common type of configuration failure has symptoms in the form of *"it worked yesterday, but it doesn't work today."* These failures are often caused by modifications to configuration data that either are not intentionally made by the users or are side effects of some intentional configuration tasks not well-understood by the users. For example, downloading a Web component, installing an application, hardware or device driver, executing a normal or malicious program, accidentally clicking mouse buttons, lending the machine to another user, etc. can potentially result in such modifications. Configuration failures are particularly hard to diagnose when they are caused by latent errors (e.g., the problem does not show up until the user invokes a particular application one week later) or when the faults reside in shared configuration data (e.g., network, Internet, and display settings) that are not intuitively tied to the failing applications.

When users encounter such failures, they usually perform *symptom-based troubleshooting* consisting of two phases, possibly iteratively: the *narrow-down phase*, in which they try to match the symptoms against their past experiences and perform various configuration tasks based on their knowledge of the applications in order to narrow down the problems to a small list of candidates, and then the *solution-query phase*, in which they try to search information sources such as the product-support database or the Web for information and solutions to similar existing problems. The effectiveness of this approach is highly variable because it relies on the users' interpretations and descriptions of their symptoms and their choices of search strings. It is particularly ineffective for newly released operating system versions, as it takes time to build up the necessary knowledge.

In this paper, we propose a *state-based approach* to troubleshooting configuration failures. The core idea is to

use comparison of *persistent-state checkpoints*[1] to identify the set of configuration data that have been modified since the target application was last known to be working; any potential configuration faults must be captured by this set, referred to as the *diff-set*. In the narrow-down phase, the diff-set is intersected with the trace of configuration-data accesses by the target failing execution to identify a smaller set of candidates that actually affect the execution. Furthermore, by comparing every pair of consecutive checkpoints on an ongoing basis, we can identify those configuration data that are naturally, constantly changing and so should be excluded from the candidate set as "false positives". The mechanical nature of this process allows both naïve and sophisticated users to quickly narrow down the problem to a small set of potential culprits. In the solution-query phase, the precise information gathered from the previous phase can often be used to help construct focused and effective search strings. Additionally, pre-collected static dependency information can be used to translate the low-level, user-unfriendly configuration-data names to high-level, user-friendly notions or presentations. For example, the mapping from individual configuration data names to the user interface programs that modify them can help point the users to appropriate places where they have a good chance of fixing the problem.

This paper is organized as follows. Section 2 describes the design and implementation of a state-based troubleshooter, and focuses on an important type of configuration data – those in the Windows Registry. Section 3 presents several actual configuration-failure scenarios involving commonly used applications, and evaluates the effectiveness of the proposed techniques in troubleshooting these problems. Section 4 summarizes the paper and discusses future work.

2. State-based Troubleshooter

We have implemented a configuration-failure troubleshooter for Windows XP machines. The current version focuses on the Windows Registry. The Registry plays an important role in the control and configuration of Windows-based systems. It stores both machine-wide and per-user settings in a hierarchical name structure wherein each node, referred to as a *Reg-key*, consists of a set of links to sub-keys and/or a set of named, typed values, referred to as *Reg-values*. (Alternatively, one can view Reg-keys and Reg-values as analogous to directories and files, respectively, in a file system.)

[1] Note that "persistent-state checkpoint" in this paper refers to a snapshot of the entire system's persistent data. It is different from the volatile-state checkpoints that have been the focus of the checkpointing and rollback recovery literature [EAW+02] and the per-process persistent state considered in a previous paper [WHV+95].

Our state-based troubleshooter consists of a main UI and four other components: (1) a checkpoint comparison tool, called *System Restore Diff (or SRDiff)*, for determining the diff-set between two checkpoints created by the System Restore [SR] feature; (2) a tracing tool for recording Registry accesses by program executions; (3) a state-ranking tool that maintains a dictionary of change frequencies of Reg-values to determine their relative importance for troubleshooting; and (4) an offline state-to-task mapping tool that records UI actions of configuration tasks and their associated modifications to Reg-values in a state-to-task mapping database.

A user typically starts the troubleshooting process by entering the main UI, which allows her to invoke the tracing tool to track the re-execution of the failed program and to invoke the checkpoint comparison tool by specifying the desired date/time ranges for the two checkpoints, i.e., the periods of time when the program was known to be working and broken, respectively. When the user hits the "Start Troubleshooting" button, the troubleshooter intersects the diff-set with the trace, uses the change-frequency information maintained by the state-ranking tool to score each individual Reg-value in the intersection, and produces an HTML troubleshooting report page listing those Reg-values with their corresponding scores. For each Reg-value that has an associated UI according to the state-to-task mapping database, the troubleshooter provides a link that the user can click on to replay the recorded user actions that bring up that UI. (We have conducted offline experiments to create the state-to-task mappings for most of the configuration tasks that users can perform from the Control Panel.) For each Reg-value with a score higher than a user-adjustable threshold, the troubleshooter provides two sets of links for searching existing online databases using parts of the Reg-value name as the search string: one for the product support database at http://support.microsoft.com/ and the other for the online document database at http://msdn.microsoft.com/. In the remainder of this section, we give more detailed descriptions of the implementations and uses of the first three components.

2.1. Checkpoint Comparison

Windows XP System Restore provides a persistent-state checkpointing and rollback mechanism to allow users with administrator privileges to undo recent undesirable changes by restoring the system to a previously checkpointed, working state. The System Restore feature is turned on by default, and automatically takes a checkpoint approximately every 24 hours, if the machine is running and when the system is essentially idle. Users can also manually create checkpoints at any time by invoking the System Restore UI from *Programs->Accessories->System Tools->System Restore*. Garbage collection of older

checkpoints is based on the amount of disk space available to System Restore, which is by default set to 12% of the total disk space for each disk drive.

The persistent states checkpointed by System Restore can be classified into two categories: files and the Registry, although the latter is ultimately stored in files as well. Registry information snapshots are taken at checkpoint time. In contrast, "snapshots" of files are taken using a kernel-mode file system filter driver that monitors every file operation, and saves a copy of each file away as part of the latest checkpoint just before it is about to be modified for the first time after the checkpoint. When the user invokes a rollback operation, all recorded file changes since the selected checkpoint was taken are undone, and the snapshot Registry information is then restored.

The rollback mechanism of System Restore can sometimes be used to repair configuration failures by rolling back all changed states without identifying which of them constitute the configuration faults. In practice, however, there are two problems. First, all installations of applications and drivers and all configuration tasks performed between the current time and the restored-to time will be lost. It can thus be an unnecessarily expensive way of fixing a potentially minor configuration fault. Second, in order to avoid rolling back user documents (which are unlikely to be root causes of the problems), System Restore only checkpoints and restores files with extensions in the Monitored File Extensions list [SRM]. Thus applications that require consistency across files not fully covered by the list can sometimes be broken by a restore operation[2].

Our checkpoint comparison tool, *SRDiff*, is an enhancement to System Restore that uses the checkpoints for the additional purpose of troubleshooting. It provides users with an opportunity to diagnose and fix their problems through localized changes whenever possible, before resorting to a system rollback with global side-effects. Given two selected checkpoints, *SRDiff* retrieves the checkpointed Registry information from their corresponding sub-directories under the *<disk driver letter>:\System Volume Information_restore{<some ID>}* directory[3]. Each corresponding pair of "root-key" files is mounted under the current Registry, and Registry APIs are used to recursively traverse the two hierarchies and output the differences in an XML-format diff-set file. The root keys include the two machine-wide security-related keys *HKLM\SAM* and *HKLM\SECURITY*, two machine-wide hardware/software-related keys, *HKLM\SYSTEM* and *HKLM\SOFTWARE*, and two user profile keys for each user account on the machine.

The *SRDiff* tool can also be used to compare checkpoints from two different machines. This is useful for troubleshooting *"it worked on that machine, but it doesn't work on this machine"* problems. While intra-machine checkpoint comparisons usually compare user-profile keys within the same user, cross-machine checkpoint comparisons typically involve comparing one user's information on machine A with another user's corresponding information on machine B.

2.2. Tracing

Registry tracing is accomplished by intercepting the Windows XP kernel-mode Registry API calls, monitoring which process is calling and recording their parameters. Specifically, a device driver is used to replace the function pointers maintained by the kernel for Registry calls with its own function pointers. There is an inherent trade-off involving the amount of trace information: collecting longer traces in general increases the chance of capturing the root cause, but also potentially introduces more false positives that make the root cause harder to identify. Therefore, the "scope" of tracing or, more specifically, which "execution-stage" traces are worth troubleshooting, needs to be carefully considered.

Typically, the troubleshooting process starts with a *single-executable, per-application-action trace* --- for example, the trace of *PowerPnt.exe* when the user hits the *Slide Show* button. If that fails to capture the root cause, the next step is to use an *all-executable, per-application-action* trace, which may additionally record Registry accesses by other processes on behalf of the target application. If that is still insufficient, then the configuration fault may have been accessed only during application initialization, but have a latent effect on the target action. So the next step is to restart the application and collect *single-executable* and *all-executable, per-application-run* traces. Further extensions of the scope would include restart of an application group and reboot of the entire machine.

2.3. State Ranking

To avoid overwhelming users, it is essential that the troubleshooter rank the relative importance of the Reg-values in the final report so that users can prioritize their efforts. We use two types of ranking in the current implementation. The first one is *order-ranking*: Reg-values are listed in the relative order of their appearance in the trace file and those at the top of the list are considered more important. This heuristic is based on the observation that Reg-values from the diff-set accessed earlier in the execution tend to be more critical because they are more likely to be the first execution point at which the failed

[2] Other system recovery software such as *Roxio GoBack* [RGB] checkpoints and restores all files to avoid inconsistency. But that exacerbates the first problem by requiring users to "roll forward" their documents as well as other data files used by various applications.

[3] This system directory is by default hidden and accessible only to the system account.

execution diverges from the correct one; once the execution diverges, later parts of the trace can be vastly different from those in the correct execution and so are less useful for identifying the root cause.

The second type of ranking is based on the observation that there exists a set of "active" Reg-values that get updated and queried much more frequently than the rest. Examples include usage counts, timestamps, window sizes and positions, cache-related information, MRU (Most Recently Used)-related information, etc. They will more likely appear in diff-sets and are less likely to be the root causes of configuration failures. Our approach is to apply the *Inverse Document Frequency (IDF)* technique [WMB99] from the information retrieval literature by treating the diff-set XML files as "documents" and the Reg-value names as "terms", and assigning scores to terms based on the inverse of their appearance frequency in those documents. Specifically, we generate the documents by running the *SRDiff* tool on every pair of consecutive checkpoints, and calculate the IDF score for each Reg-value V in the troubleshooter report using the following logarithmic formulation [ZM]: *IDF score for V = log (1 + N / F_V)*, where N is the total number of diff-set XML files and F_V is the number of files that V appears in. The IDF scores are displayed next to their corresponding Reg-values in the report. Reg-values with scores lower than a threshold are displayed in a light color to allow those with a higher score to stand out.

Since the set of active Reg-values depends on what software is installed, what applications are running, and the user's usage pattern for those applications, the *diff-set* documents ideally should be collected on a per-machine basis. To simplify installation, our current tool includes a static dictionary generated on a particular machine. We have observed that it is effective in noise-filtering common Windows XP Registry entries.

3. Experimental Results and Case Studies

It is well known that the Windows Registry contains vast amounts of configuration-related data. Machine statistics such as the total number of Reg-values, how many of them change, and how often, depend on the set of software installed on a particular machine and the user's usage pattern. A preliminary survey of eight machines indicates that the total number of Reg-values can range from 150,000 to 300,000; the number of Reg-values changed per day can fluctuate significantly between tens (e.g., when the machine is not used over the weekend) and tens of thousands (e.g., when a major software package or patch is installed), and typically ranges from hundreds to a few thousand. The major challenge for the state-based troubleshooting approach is to single out the root cause from the huge set of Reg-values.

On an author's desktop machine, there were 88 checkpoints spanning a period of just under 3 months. The diff-set between the earliest and latest checkpoints contained 48,132 Reg-values. The 87 pair-wise diff-set files contained approximately 60,000 unique Reg-values. By setting the IDF frequency threshold to be 10% (the value we used to produce the results presented in the remainder of this section), we found about 1,200 Reg-values to be, by that standard, frequently-changing. Although they constituted only two percent of the 60,000 Reg-values, these 1,200 values would have been highly likely to appear in any diff-set, and hence in the troubleshooter report as false positives.

We next describe four actual user scenarios to evaluate the effectiveness of our troubleshooter. Four numbers are of interest: the numbers of Reg-values in the *diff-set* (N_d), in the intersection of the *diff-set* and the trace (N_T), and in the above-the-IDF-threshold set (N_I); and the order-ranking (N_o) of the root-cause Reg-value among the N_I values. Table 1 summarizes the numbers from the four case studies. It illustrates the capability of our combination of techniques to reduce the number of "interesting" Reg-values by orders of magnitude.

Table 1. Summary of Case Studies.

	N_d	N_T	N_I	N_o
PowerPoint	2,015	2	2	1
IM	2,329	21	9	1
Media Player	72,656	12	12	1
Word	1,989	4	1	1

3.1. PowerPoint *Slide Show* Problem

Scenario: Steve purchased a new laptop a few months ago, and had been able to run PowerPoint slide shows on the laptop without any problem. During a trip last week, he let a colleague borrow the laptop. Today, when he tried to start a slide show, it was no longer working: he hit the slide show button and nothing appeared on the screen, except that a minimized window was added to the taskbar. He rebooted the machine and it didn't help.

Troubleshooting: Steve first invoked the troubleshooter to record the Registry access trace for PowerPoint when he hit the slide show button. Then he invoked *SRDiff* by specifying two date ranges before and after the trip (for the first and second checkpoints, respectively). The two selected checkpoints were 12 days apart and the intersection contained only two Reg-values, both of which have the same UI link pointing to the *Control Panel->Display->Settings->Monitor 2* page. Steve noticed that the checkbox *"Extend my desktop to this monitor"* was selected. Since he had never used a second monitor with his laptop, he cleared that checkbox and found that the slide show problem was fixed.

Root cause: Windows XP supports a dual-monitor mode to allow users to increase their desktop space by

attaching a second monitor. Steve's new laptop came with a video card that can support that mode, and his colleague apparently enabled that mode. When the dual-monitor mode is turned on, PowerPoint will by default send the slide show to the video output for the second monitor, even if there is no such monitor physically attached. This unfortunately created a confusing experience for Steve, who was never even aware of the dual-monitor mode. The Reg-value that indicates the dual-monitor mode is a machine-wide setting stored under the *HKLM\SYSTEM* root-key.

Effectiveness of the tool: In this example, a single-executable, per-action trace (with only 268 lines) was sufficient to narrow down the problem. The tool showed N_d = 2,015 and N_T = 2, which demonstrated that the intersection of the diff-set with the execution trace was very effective in reducing the number of candidates. The IDF scoring was not necessary in this case, and the state-to-task mapping suggested the correct configuration task to fix the problem.

3.2. Instant Messaging (IM) Performance Problem

Scenario: Jamie had been running Windows Messenger on a corporate machine to communicate with her MSN IM buddies out on the Internet without any problem. Lately, however, she observed that the responsiveness of the IM service had degraded seriously -- so much so that she could no longer have reasonable real-time communications with her friends. She suspected that the degradation was due to network congestion, either in the corporate network or the Internet, or because of an overloaded IM server. But the problem persisted for many days and none of her buddies seemed to observe the same problem.

Troubleshooting: Jamie first used the troubleshooter to record only the trace when she hit the *"Send"* button on an IM window. She manually created a checkpoint and invoked *SRDiff* to compare the new checkpoint with another one taken six days previously. Unfortunately, the report did not reveal any useful information. She then exited Windows Messenger and recorded another longer trace that covered the entire process of starting the application, logging on to the IM service, and sending a message. This time, the first above-the-threshold entry pointed to the *Control Panel->Firewall Client Options* page, which showed that the firewall client software had been disabled. Jamie enabled the software and that fixed the performance problem.

Root cause: Firewall client software must be enabled to allow Windows Messenger to communicate with the IM service through a corporate firewall proxy server. Apparently, Jamie disabled the software when she was troubleshooting another network-related problem the previous week. Without the firewall client running, Windows Messenger switched to an HTTP polling mode with a polling frequency of 20 seconds, which allowed the IM to continue providing service but with greatly degraded performance.

Effectiveness of the tool: In this example, a per-action trace was insufficient and a per-run trace (with 6,975 lines) was necessary because the accesses to firewall client-related Reg-values only occurred during the start-up and log-on phase of the program's execution. The statistics were N_d = 2,329, N_T = 21, and N_I = 9, showing that IDF scoring effectively reduced the size of the candidate set by (21-9)/21=57%. More importantly, it moved the order-ranking of the root-cause Reg-value from No. 4 among the 21 values to No. 1 among the 9 values, and quickly led the user to the correct configuration task page to fix the problem.

3.3. Media Player *"Open URL"* Problem

Scenario: Ken had been using the Media Player to listen to music and online presentations without any problem. One day, when he was trying to play a recorded presentation by invoking the *"Open URL"* function to connect to an intranet server, he received an error message asking him to "*connect to the Internet"*. Since Ken could still browse Web pages on both the Internet and the intranet, he suspected that the intranet media server was down. But when he asked his colleague Andrea to try it, she was able to play that presentation with exactly the same version of Media Player.

Troubleshooting and root-cause: Ken used the troubleshooter to record a trace between hitting the "OK" button on the *"Open URL"* window and receiving the error message. He invoked *SRDiff* to compare the two most recent checkpoints on his machine and Andrea's machine. The first entry in the report, named *"Internet Settings\EnableAutodial"*, pointed to the *Control Panel->Internet Options->Connections* page that contained selections of auto-dial-up settings but were unfortunately grayed out and not available on Ken's machine. By clicking on the accompanying search link for the knowledge base, Ken saw an article that explained the meaning of the *"EnableAutodial"* Reg-value and mentioned that some application might erroneously set the value to 1. Since Ken's machine is always connected to the Internet through the corporate network and should never dial a connection, he changed the value to 0 and that solved the Media Player problem.

Effectiveness of the tool: In this example, the cross-machine checkpoint comparison was performed in two steps: first, the corresponding machine-wide root keys were compared, as in the intra-machine case; then, Ken's user-profile keys on his own machine were compared with Andrea's corresponding keys on her machine. The root cause was found among the latter. The resulting *diff-set* had N_d = 72,656, which was more than an order of magnitude larger than the corresponding figure for the

previous same-machine, cross-instance use. But taking its intersection with the 2,549-line trace significantly reduced the candidate set size to N_T = 12. IDF scoring did not identify any of them as false positives in this case. The UI link gave good hints as to what that Reg-value might mean, but could not be used to solve the problem because the availability of certain configuration options depends on each individual machine's settings. Online database searches with the target Reg-value name proved to be effective; vague search strings such as "Media Player connection problem" or parts of the error message would have led to a large number of unrelated articles.

3.4. Microsoft Word *"Preparing to install"* problem

Scenario: Mike had been using Word for writing documents. One day, he suddenly started having the following problem: every time he launched Word, a *"Preparing to install"* dialog box popped up, followed by a long installation of some software. Since he did not ask for the installation of any new features, he always hit the "Cancel" button and was able to continue using Word normally. But the problem persisted.

Troubleshooting and root cause: Mike recorded the Word-launching trace and ran *SRDiff* on the most recent checkpoint and an older checkpoint taken five days previously. The first Reg-value in the intersection pointed to a Reg-key shared by all Office applications, and had a name ending with *"\LanguageResources\1041"*. Database searches based on that name revealed that 1041 is the Locale Identifier for Japanese, and, since the data of the Reg-value was *"On"*, Word was apparently trying to install the Japanese language support, probably as a result of Mike's unintentionally trying to open a Japanese document a few days previously. Since Mike cannot read Japanese at all, he turned off the Reg-value and thereby fixed the problem.

Effectiveness of the tool: The troubleshooter report had N_d = 1,989, N_T = 4, N_I = 1, and N_o = 1. The IDF scoring effectively filtered out three of the four candidates as false positives. More importantly, two of them had the string *"installer"* as part of their names, and could have misled the troubleshooting process. Apparently, these were usage counts of some sort that had been constantly changing and hence had very low IDF scores.

4. Summary, Limitations, and Future Work

We have proposed a state-based troubleshooting framework for diagnosing application failures caused by configuration errors, and presented the design and implementation of a state-based troubleshooter for the Windows Registry. Preliminary results from four case studies showed that the approach is very promising in providing effective troubleshooting of configuration failures. Experimental results from a large number of real-world cases would be required for a comprehensive evaluation of the techniques.

A major limitation of the current troubleshooter is its lack of support for handling *indirect, asynchronous dependencies*. Specifically, a service/daemon may read a large number of Registry entries during the booting or log-in process, and "cache" them in volatile variables. Some time later, an application may invoke functions exposed by the service/daemon and fail due to erroneous values of some of those volatile variables. In such a case, the root-cause Registry entry is read by the service/daemon before the application is even started, and so the application trace would not capture the root cause. Indirect, asynchronous dependencies introduce several challenges. For example, there needs to be a process dependency tracking mechanism so that, after the troubleshooter fails to find the root cause from the application trace, the user is told to expand the scope of tracing by restarting the service/daemon as well. Another challenge is that such a trace is usually much larger than a single-application trace and can potentially introduce a large number of false positives. More sophisticated statistical analysis or more fine-grained dependency tracking may be necessary for narrowing down the candidate set. We plan to investigate these issues in our future work.

References

[EAW+02] E. N. Elnozahy, L. Alvisi, Y. M. Wang , and D. B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-passing Systems," *ACM Computing Surveys*, Vol. 34, Issue 3, pp. 375 – 408, Sept. 2002.

[GR93] J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques," Morgan Kaufmann, 1993.

[RGB] Roxio GoBack 3 System Recovery Software, http://www.roxio.com/en/products/goback/index.jhtml.

[SR] Windows XP System Restore, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwxp/html/windowsxpsystemrestore.asp.

[SRM] System Restore Monitored File Extensions, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sr/sr/monitored_file_extensions.asp.

[SS92] D. P. Siewiorek and R. S. Swarz, "Reliable Computer Systems: Design and Evaluation," Second Edition, Digial Press, 1992.

[WHV+95] Y. M. Wang, Y. Huang, K.-P. Vo, P. Y. Chung, and C. Kintala, "Checkpointing and Its Applications," in *Proc. IEEE Fault-Tolerant Computing Symposium (FTCS-25)*, pp. 22-31, June 1995.

[WMB99] I. H. Witten, A. Moffat, and T. C. Bell, "Managing Gigabytes", Morgan Kaufmann Publishers, Inc., Second Edition, 1999.

[ZM] J. Zobel and A. Moffat, "Exploring the Similarity Space", http://goanna.cs.rmit.edu.au/~jz/fulltext/sigirforum98.pdf.