Why Did My PC Suddenly Slow Down?

Sumit Basu
sumitb@microsoft.com

John Dunagan
jdunagan@microsoft.com

Greg Smith
gregsmi@microsoft.com

Microsoft Research, One Microsoft Way, Redmond, WA 98052

ABSTRACT

Users are often frustrated when they encounter a sudden decrease in the responsiveness of their personal computers. However, it is often difficult to pinpoint a particular offending process and the resource it is over-consuming, even when such a simple explanation does exist. We present preliminary results from several weeks of PC usage showing that user-perceived unresponsiveness often has such a simple explanation and that simple statistical models often suffice to pinpoint the problem. The statistical models we build use all the performance counters for all running processes. When the user expresses frustration at a given time point, we can use these models to determine which processes are acting most anomalously, and in turn which features of those processes are most anomalous. We present an investigative tool that ranks processes and features according to their degree of anomaly, and allows the user to interactively examine the relevant time series.

KEYWORDS: performance instrumentation, machine management, statistical modeling, anomaly detection.

1. INTRODUCTION

Nearly everyone who has used a computer has encountered a situation where an application or the entire machine seems to slow down dramatically: all of a sudden, windows are not as responsive, actions are taking longer than they should, and so on. At this point, although the user might like to investigate what's wrong, he has a limited set of options. He may open up Windows' Task Manager or use UNIX's "ps" command to view the running processes, and then check to see which ones are taking the most CPU, I/O, or memory, but he will generally not know whether the values he sees are typical or surprising. In other words, while he can view instantaneous values of some system features, he has no model of what their typical values are; furthermore, even if he could view all possible features, it would be difficult to glean insights from the resulting deluge of data.

One plausible hypothesis is that most slowdowns are the result of one process consuming an abnormally high amount of one resource (e.g., CPU, disk, network, OS handles or file descriptors, system threads, etc.) from a large set of possible resources. However, presenting the consumption of every resource for every process directly would likely be too much information for a user to digest. We built a system based on these assumptions, and found it to be remarkably effective. Our system collects data and builds a model for each process. The model allows us to determine the level of anomaly for any process, and furthermore for any feature within a process. This allows us to use all possible features, since only the anomalous ones will float to the top. We have also developed a visualization tool that allows interactive investigation of the processes and their features with respect to these models. When a user experiences a slowdown at a given point, he can see the processes ranked by their relative level of anomaly, and for each process, the features ranked by anomaly. In addition, the user can see the time series for the feature of interest.

Once a user has identified a high-likelihood offender (perhaps an antivirus product or a desktop search application), he has numerous options to improve the situation. He might start shopping for a new antivirus product, switch to a competing desktop search application, or just stop using something that is more trouble that its worth. Additionally, because many developers also use the software they write, this tool may help them catch transient resource usage issues that are significantly slowing down the system as a whole.

The primary question raised by our approach is its effectiveness: how often does user-perceived machine slowness have such a simple explanation? Beyond this, there were also significant questions about how best to represent features that would only be available sporadically (i.e., when the relevant processes were running). In the remainder of this paper, we address these questions and show preliminary results from using our investigative tool.

2. RELATED WORK

There has been a significant amount of work using statistical models to detect and/or diagnose faults and performance problems [A+03, Ba+04, Bo+05, Ch+02, F+04, G+05, KF05, R+04, R05, X+04]. Researchers have investigated many different sources of data (e.g., performance counters [Co+04], request paths [Ch+03, Ch+04]), as well as many different statistical models. Most of this work has focused on a server environment:

an environment where a large number of machines serve an even larger number of user requests. In addition to their obvious economic importance, the high request volume of server environments makes them particularly well-suited to analysis using statistical models. The consumers of this analysis are either operations personnel (sometimes viewed as datacenter system administrators) or developers.

In contrast to this previous line of research, our work focuses on end-user desktops, and the consumer of the analysis is the end-user himself. End-user desktops are a significantly different environment from servers. Perhaps most importantly, we have much less expectation of the workload being repetitive. Because of this, we might find it quite difficult to duplicate the success that statistical models have had detecting request failures in server environments. Luckily, we can sidestep this issue because the consumer of the analysis is the frustrated user – the user will himself indicate that the system is slow (detection), and the statistical model is only responsible for narrowing down the reason for this slowness (diagnosis).

Like our work, statistical debugging [Z+04, Z+05, Z+06], Strider [Wa+03], Chronus [Wh+04] and Peer-Pressure [Wa+04] target end-user applications, but they are otherwise radically different. Statistical debugging focuses on helping developers understand why a particular application occasionally fails. To this end, statistical debugging requires an external mechanism to determine which process is failing (in contrast to our goal of determining which process is at fault), and it requires a large number of differently-instrumented binaries (in contrast to our goal of working on a single machine without some external correlation mechanism and without instrumentation). Strider, Chronus and Peer-Pressure aim to diagnose problems due to bad persistent state (i.e., file contents and registry settings). Strider and Peer-Pressure presume that the failing process is known a-priori, while Chronus requires the user to specify a probe determining if the failure is present. In contrast, our analysis is not restricted to persistent state changes (some other input or change in workload may have triggered resource over-consumption), and we require significantly less expertise from the user (he just pushes the "why is my machine slow" button).

There has also been a significant amount of previous work on helping developers or sophisticated system administrators understand performance of individual components on a single machine, which may be either a server or an end-user's desktop. For example, profilers and other instrumentation or logging systems allow a developer or sophisticated system administrator to understand where the time/memory/etc. is spent in an application or OS, thereby guiding refinements to the application or OS [Ca+04, HC+01]. Our work relies on Windows performance counters (one such logging system) to gather the low-level performance metrics; our contribution is to present more useful analysis of this data to the end-user. We are not aware of previous work trying to make the analysis done by these systems useful to an end-user.

A final distinction with previous work is our focus on the user's *perception* of slowness. In contrast, most previous work has looked at more directly measureable quantities, like the latency of a particular machine operation.

3. DATA COLLECTION AND MODELING

To evaluate our approach, we collected data over the course of several weeks on two machines running Windows XP. Every time a user of either machine was frustrated at the slowness of the system, he would press the "why is my machine slow" button (Scroll Lock in our implementation), which we refer to as a "frustration event". During this time, there was approximately one frustration event per day per machine.

Instead of carefully choosing which features should be used for the analysis, we gathered all available performance counters for all processes. This included items such as total number of threads in the process, I/O bytes read per second, percentage of CPU used, page faults, and so on. Our reasoning was that more data could only help us, since we would be prioritizing features to show the user the ones that were most relevant – as such, if we included uninteresting or constant features, they would simply sink to the bottom of the list.

These features were written to a file for all running processes once every 60 seconds. Every four hours, the resulting file was compressed and copied to a fileserver. We found the overhead of this logging to be negligible (in particular, on a 3Ghz Xeon PC, less than 0.15% of the CPU on average).

The nature of the data is complex – not only are there dozens of features for each of dozens of processes; at any given time only some processes are running. Figure 1 below illustrates this aspect of the data.

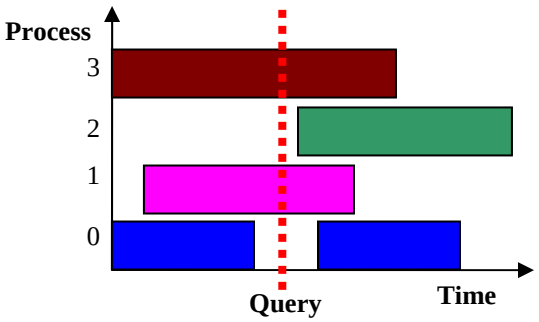

Figure 1: Process lifetimes shown against time. At the query time, only a subset of the processes are running.

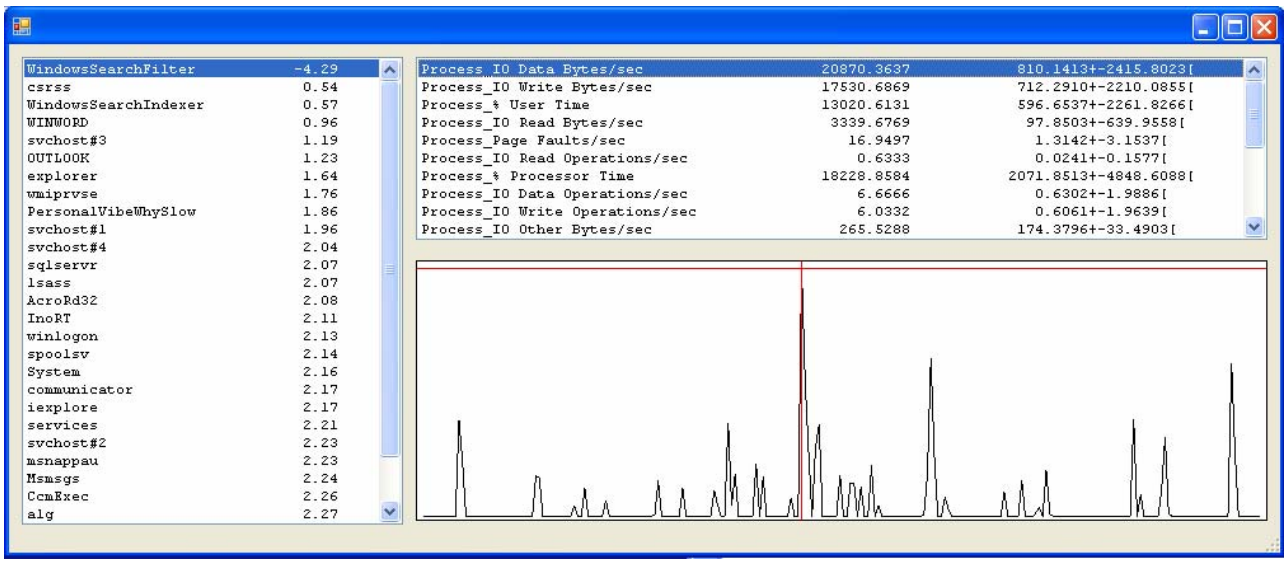

Figure 2: The Investigative Tool. The left pane shows per process average likelihoods; the upper right pane shows the non-constant features for that process ranked by anomaly; the lower right pane shows the time series.

Our dataset is thus *tree-structured*: the first level nodes are processes, whose children are the features for that process; each of those features contains a time series which may have large gaps and may not match up with other features/processes, as illustrated in Figure 3.

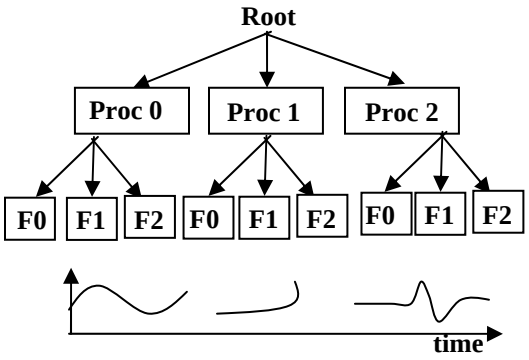

Figure 3: Tree-Structured Dataset. Each process has multiple features, and each feature has a time series which may be non-contiguous in time.

When the user makes a query at a particular point in time, only some of the processes are running. This results in an instantaneous sample which is also tree-structured. In order to extract this sample, we traverse the dataset and see if there is a value in the time series for each feature (leaf node) that is within a window of the query time, and then keep the closest such value.

For this initial study, we chose to model each feature independently as a univariate Gaussian. Though we know our features will not all be independent of each other (for instance, I/O total bytes and I/O read bytes are far from independent), this makes dealing with missing data far easier. We trained the models independently over each 4-hour window corresponding to a particular log file, and then used this model for any frustration event in the same 4-hour window. Though this was sufficient for our investigation of historical data, fully supporting the interactive scenario outlined in the Introduction will obviously require looking only at data gathered prior to a frustration event – this is one of our tasks for future work.

To compute the log likelihood of a given process, we could simply take the sum of the log likelihoods of the individual features. Unfortunately, features that are constant in a particular process lead to numerical instability in this computation, as the empirical variance will be zero. After excluding constant features, we need a different method to compare different processes with different numbers of features. We thus compute the *average likelihood per feature* for each process that has more than a minimal number of non-constant features (10 in our experiments).

4. THE INVESTIGATIVE TOOL

In order to make this data accessible to the user, we developed an interface to help the user peruse the performance data (see Figure 2 above). The left pane shows a list of processes ranked by their level of anomaly; the upper right pane shows the features for that process along with the feature values as compared to the mean and standard deviation, again ranked by anomaly. The lower right pane shows a plot of the time series for that feature, and the red crosshairs pinpoint the query time and the feature value. We feel that it is important for the user to be able to see the entire time series – this gives his context about how atypical the value is, whether it was a sudden transition or a slow climb/fall, whether this kind of behavior occurs regularly, etc.

At present, the UI only consumes performance data that has been logged to disk. We plan to remove this limitation soon so that the UI can appear at the moment the user presses the "why is my machine slow" button, completing the scenario we described in the Introduction.

5. RESULTS

To investigate the effectiveness of our proposed approach, we looked at the collection of time points at which the user hit the "why is my machine slow" button. Over the course of 53 machine-days of data from two users, there were 36 such occasions, of which five were duplicates for our purposes (the user pressed the button more than once within a few seconds). For each of the remaining 31 cases, we examined the data point with respect to its four-hour context in the investigative tool as described above. Our first evaluation was determining the fraction of query points for which the tool correctly identified the process and features which were causing the slowness. Of course, we cannot know for certain what the cause was, but the presence of large spikes in CPU, I/O, memory, and/or other resources made it very plausible in all but 3 cases that a given process (or set of processes) was causing the problem.

Table 1 below examines whether the top process identified by our investigative tool was a plausible cause of the slowness:

Top process is plausible source of slowness	26 / 31 (83.9%)
One of top two processes is plausible source	28 / 31 (90.3%)
Source of slowness unclear from tool	3 / 31 (6.4%)

Table 1: Performance for identifying the process causing slowness.

In two of the cases where the source of slowness was unclear, no process that we could determine had unusual features. It is possible that the user hit the frustration button by accident, but since we currently do not have user annotations of these events, we cannot be sure. The third case is described in case study 3 below.

For the 26 cases in which the process was correctly identified, Table 2 considers whether the top feature identified by our tool was a plausible cause of the process behavior:

Though these results are promising, we were curious whether CPU load alone could determine the cause of slowness. If so, Task Manager or ps would be sufficient to find the answers. Our experience has led us to believe CPU load is not always the answer, and with the data in hand, we looked at what the plausible primary and contributing factors for slowdown were in each case. Table 3 below shows the results for the 25 cases where we were able to identify highly plausible causes for the slowness:

Top feature is plausible source of slowness	25 / 26 (96.2%)
Source of slowness unclear	1 / 26 (3.8%)

Table 2: Performance for identifying features causing slowness.

	CPU	Mem	I/O	Handle Count	Page Faults	Thread Count
Most anomalous feature	8/ 25	4/ 25	11/ 25	1/ 25	1/ 25	0/ 25
Contributing feature	14/ 25	7/ 25	16/ 25	2/ 25	7/ 25	1/ 25

Table 3: Breakdown of primary and contributing factors for slowness by CPU, Memory, IO, Handle Count, Page Faults, and Thread Count.

In these preliminary results, IO-related features most often appeared as the most anomalous feature, though CPU was close behind. Furthermore, in a handful of cases, handle counts, page faults, and thread counts appeared to be primary factors or contributors to the slowness. This is particularly interesting in that tools like Task Manager and ps are typically used only to examine CPU consumption.

To give the reader a better sense for the results, we now go into three case studies that describe three different modes of behavior: a case with an obvious single process causing slowness, a case with multiple processes interacting to cause slowness, and a case where our method completely fails to identify the source of the problem (but we have a good guess as to what happened).

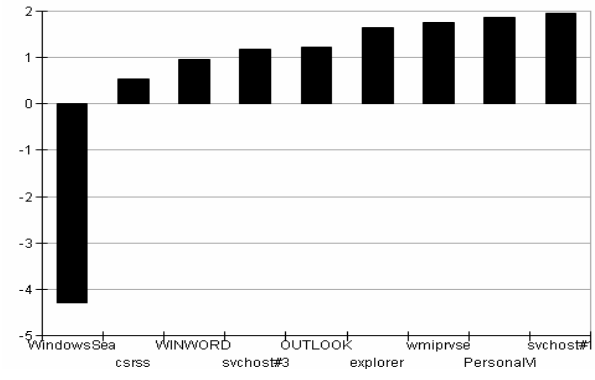

Figure 4: Average per-feature log-likelihood for the top ten processes in case study 1.

Case Study 1: One Process Causing Slowness

In this instance, the WindowsSearchFilter ("WindowsSea" in the figure) process has a far lower average log-likelihood than the other processes, as we can see in Figure 4 above. In this case, then, it is clear in the tool which process is likely to be at fault. The top feature (i.e., the one with the lowest likelihood) for this process is "IO Data Bytes/sec," and we show the time series in Figure 5 below.

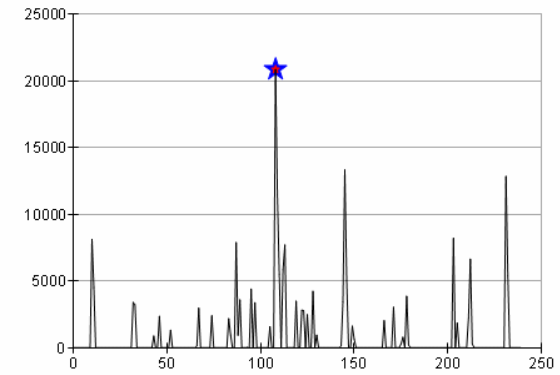

Figure 5: The top ranked feature, IO Data Bytes/sec, for process WindowsSearchFilter in case study 1. The star marks value at the user query time.

In this case, not only does the feature have a peak at the query time, it is also the highest value in the four hour contextual window. As such, the investigation tool has showcased the plausible cause of the slowness via its top-ranked process and feature; without any further clicking the user would have a good guess of what's going wrong.

Case Study 2: Multiple Processes Involved

The second case is more complex. We can see this when we look at the likelihood distribution over the top ten processes in Figure 6 below:

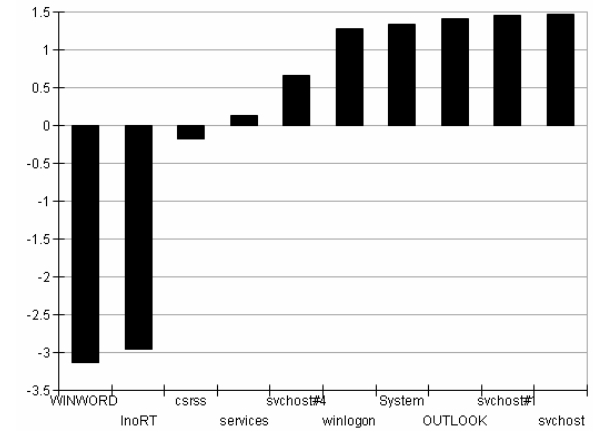

Figure 6: Per-process average likelihoods for case study 2.

Both WINWORD and InoRT (an anti-virus program) are strongly anomalous in this case. When we examine the features in detail, we find that both are jumping in terms of memory usage. It seems plausible that the virus checker is intercepting and checking a large file as Word tries to open it; this is causing both to increase their memory footprint dramatically.

Case Study 3: A Failure Mode

For our final example, we look at a case where things did not work so well. The distribution of process likelihoods implies that something might be amiss (see Figure 7 below), since many processes seem to be anomalous:

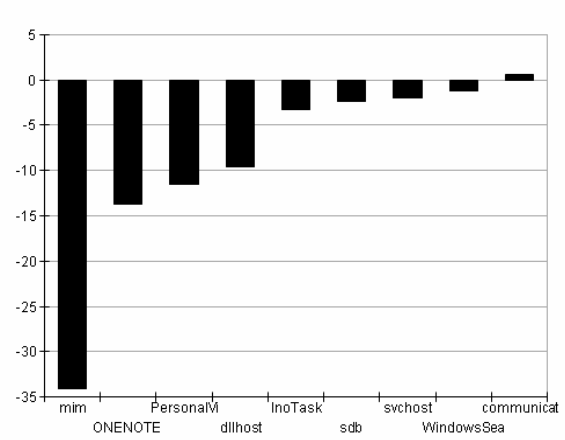

Figure 7: Per-process average likelihoods for case study 3.

This puzzled us initially, but once we examined the features, we saw that the frustration point was at the beginning of each time series, and that all the features for each process quickly ramped up to their usual states. Thus this first point in the time series seemed anomalous to our model, since all of the values were far below usual. What seems likely to have happened in this case is that the machine had just restarted, and the user was frustrated waiting for the system to become responsive. We consider this a failure for our method, since none of our indicators help explain why the system is being slow during startup (e.g.., is it paging in code, spending CPU running a startup script, or something else entirely?) or even identify which process (if any) is causing the most problems. We hope that incorporating system-level features may help diagnose this kind of situation in our future work.

6. DISCUSSION AND FUTURE WORK

Our initial experimental results are quite promising. Our immediate work plan is to make the interactive version of our tool fully functional. Beyond this, there are many interesting open questions.

First, we have not yet been able to validate the causal relationships we believe we have uncovered. In cases where a process that the user is not actively engaged with causes the rest of the system to slow down, simply rate-limiting the aggressive process may be enough to maintain the user's perception of overall system responsiveness. If such a technique were successful, it would provide empirical validation that the process and resource identified as anomalous were indeed at fault (in addition to improving the overall user experience). The case where the process at fault is one that the user is actively engaged with seems harder, and it may be necessary in this case to involve a developer for the relevant application to validate the diagnosis (and possibly to fix the problem as well). Developing and evaluating some validation strategy is one of our areas for future work.

Second, there are obvious opportunities to employ more sophisticated models and additional data sources. One we are particularly excited about is allowing the user to optionally annotate frustration events at the time they occur. Mapping such annotations to the machine-level symptoms might reveal interesting new connections and allow more detailed analyses.

Third, we have not yet undertaken a user study with the tool. In the future we would like to evaluate whether particular classes of users consider the tool to be useful, whether they decide to make changes in their software usage because of the tool's output, and whether these changes result in fewer incidents of sudden system slowdown.

7. REFERENCES

[A+03] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. "Performance debugging for distributed systems of black boxes." In SOSP 2003.

[Ba+04] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. "Using Magpie for request extraction and workload modelling." In OSDI 2004.

[Bo+05] P. Bodik, G. Friedman, L. Biewald, H. Levine, G. Candea, K. Patel, G. Tolle, J. Hui, A. Fox, M. I. Jordan, and D. Patterson. "Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization." In ICAC 2005.

[Ca+04] B. M. Cantrill, M. W. Shapiro and A. H. Leventhal. "Dynamic Instrumentation of Production Systems." In USENIX Annual Technicla Conference 2004.

[Ch+02] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. "Pinpoint: Problem Determination in Large, Dynamic Internet Services." In DSN 2002.

[Ch+03] M. Y. Chen, E. Kiciman, A. Accardi, A. Fox, and E. Brewer. "Using Runtime Paths for Macro Analysis" In HotOS 2003.

[Ch+04] M. Y. Chen, A. Accardi, E. Kiciman, A. Fox, D. Patterson, and E. Brewer. "Path-Based Failure and Evolution Management." In NSDI 2004.

[Co+04] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase. "Correlating instrumentation data to system states: A building block for automated diagnosis and control." In OSDI 2004.

[F+04] A. Fox, E. Kiciman, D. Patterson, M. Jordan, and R. Katz. "Combining Statistical Monitoring and Predictable Recovery for Self-Management." In 2004 Workshop on Self-Managed Systems (WOSS'04) in conjunction with ACM SIGSOFT FSE-12.

[G+05] M. Goldszmidt, I. Cohen, A. Fox, and S. Zhang. "Three research challenges at the intersection of machine learning, statistical induction, and systems." In HotOS 2005

[HC01] M. Hirzel and T. M. Chilimbi. "Bursty Tracing: A Framework for Low-Overhead Temporal Profiling." In ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO), 2001.

[KF05] E. Kiciman and A. Fox. "Detecting application-level failures in component-based internet services." In IEEE Transactions on Neural Networks, Spring 2005.

[R+04] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik. "Real-time Problem Determination in Distributed Systems using Active Probing." In NOMS 2004.

[R05] I. Rish. "Distributed Systems Diagnosis Using Belief Propagation." In Allerton Conference on Communication, Control and Computing, 2005.

[Wa+03] Y. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang. "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support." In LISA 2003.

[Wa+04] H. J. Wang, J. Platt, Y. Chen, R. Zhang, and Y. Wang. "Automatic Misconfiguration Troubleshooting with PeerPressure." In OSDI 2004

[Wh+04] A. Whitaker, R. S. Cox, and S. D. Gribble. "Configuration Debugging as Search: Finding the Needle in the Haystack." In OSDI 2004.

[X+04] W. Xu, P. Bodik, and D. Patterson. "A Flexible Architecture for Statistical Learning and Data Mining from System Log Streams." In workshop on Temporal Data Mining: Algorithms, Theory and Applications at ICDM 2004.

[Z+04] A. X. Zheng, M. I. Jordan, B. Liblit, and A. Aiken. "Statistical debugging of sampled programs." In NIPS 2004.

[Z+05] S. Zhang, I. Cohen, J. Symons, and A. Fox. "Ensembles of Models for Automated Diagnosis of System Performance Problems." In DSN 2005.

[Z+06] A. Zheng, M. I. Jordan, B. Liblit, M. Nayur, and A. Aiken. "Statistical debugging: Simultaneous identification of multiple bugs." In ICML 2006.