PeerPressure for Automatic Troubleshooting

Helen J. Wang, John Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang

Microsoft Research

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software**]: Miscellaneous

General Terms

Design, Experimentation, Measurement

Keywords

System Management, Automatic Troubleshooting, Golden State, Statistics, Bayesian Estimates, PeerPressure

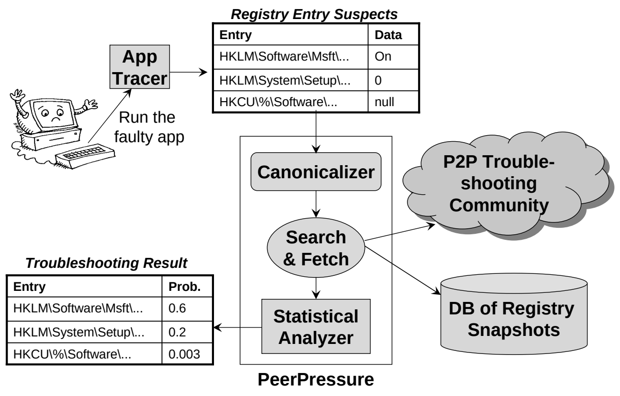
1. INTRODUCTION

Technical support contributes 17% of the total cost of ownership of today's desktop PCs [3]. An important element of technical support is troubleshooting misconfigured applications. Misconfiguration troubleshooting is particularly challenging, because configuration information can be shared and altered by multiple applications. Maintaining healthy configurations of a computer platform with a large installed base and numerous third-party software packages has been recognized as a daunting task [1]. The considerable number of possible configurations and the difficulty in specifying the "golden state" [4], the perfect configuration, have made the problem appear to be intractable.

In this paper, we address the problem of misconfiguration troubleshooting. There are two essential goals in designing such a troubleshooting system:

1. Troubleshooting effectiveness: the system should effectively identify a *small* set of sick configuration candidates with a short response time;
2. Automation: the system should minimize the number of manual steps and the number of users involved.

To diagnose misconfigurations of an application on a sick machine, it is natural to find a healthy machine to compare against [7]. Then, the configurations that differ between the healthy and the sick are misconfiguration suspects. However, it is difficult to identify a healthy machine *automatically*. Involving the user in confirming the correct application behavior seems unavoidable



Entry	Data
HKLM\Software\Msft\...	On
HKLM\System\Setup\...	0
HKCU\%\Software\...	null

Entry	Prob.
HKLM\Software\Msft\...	0.6
HKLM\System\Setup\...	0.2
HKCU\%\Software\...	0.003

Figure 1: PeerPressure Troubleshooter Architecture and its Operations

We avoid this extensive manual identification work by observing that *the golden state is in the mass*. In other words, an application functions correctly on *most* of machines, therefore we can use the statistics from a large enough sample set as the "statistical golden state". The statistical golden state can be combined with Bayesian statistics to identify anomalous misconfigurations on sick machines. Then, the misconfigurations can be corrected by comforming to the majority of the samples. We name this statistical troubleshooting method *PeerPressure*.

To simplify our presentation, we focus our discussion on a particular type of important configuration data, the Windows Registry [2], which provides hierarchical persistent storage for named, typed entries. The principles and techniques are directly applicable to other types of configuration stores such as files and other platforms such as Unix.

2. PEERPRESSURE ARCHITECTURE

Figure 1 illustrates the architecture and the operations of a PeerPressure troubleshooting system. A troubleshooting user first expresses the symptom of the sick machine through the use of "App Tracer". "App Tracer" records the registry entries that are used as input to the failed application execution. We term these misconfiguration candidates *suspects*. Then, the user feeds the suspects into the PeerPressure troubleshooter which has three modules: a canonicalizer, a searcher/fetcher, and a statistical analyzer. The canonicalizer turns any user- or machine-specific entries into a *canonicalized* form. For example, user names and machine names are all replaced with constant strings "USERNAME" and "MACHINENAME", respectively. Then, PeerPressure searches for a sample set of machines that run the same application. The search can

	Name	Suspect	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5
e1	.jpg/contentType	*image/jpeg*	image/jpeg	image/jpeg	image/jpeg	image/jpeg	image/jpeg
e2	.htc/contentType	not exist	text/x-comp	text/x-comp	text/x-comp	text/x-comp	text/x-comp
e3	url-visited	yahoo	hotmail	nytimes	SFGate	google	friendster

Table 1: Intuition behind PeerPressure Sick Probability Formulation

be performed over a database of machine configuration snapshots or through a peer-to-peer troubleshooting community. (In this paper, we base our discussions on the GeneBank database approach. For the peer-to-peer approach, we refer interested readers to [6].) Next, PeerPressure fetches the respective values of the canonicalized suspects from the sample set machines. The statistical analyzer then performs statistical analysis, calculates the probability for each suspect to be sick, and outputs a ranking report based on the sick probability. Finally, PeerPressure conducts trial-and-error fixing, by stepping down the ranking report and replacing the possibly sick value with the most popular value from the sample set. The fixing step interacts with the user to determine whether the sickness is cured. This last step is not shown in the figure; and we will not further address it for the rest of the paper.

3. THE PEERPRESSURE ALGORITHM

We use an example to illustrate the intuition and objectives of formulating the sick probability calculation for each suspect. Table 1 shows three suspects ($e1$,$e2$,$e3$) and their respective values from a sample set of machine configuration snapshots from our database. A cursory examination of the sample set suggests that $e1$ is probably healthy; $e2$ is more likely to be sick than $e3$ because all samples have the same value, while the suspect value is different.

In fact, we have seen two types of state in canonicalized configuration entries: (I) application configuration states such as $e1$ and $e2$, (II) operational states such as timestamps, usage counts, caches, seeds for random number generators, window positions, and MRU (Most Recently Used)-related information. For troubleshooting configuration failures, we are mostly concerned with type I entries. Type II entries constitute the “natural biological diversity” among machines and are less likely to be root causes of configuration failures. In our example, $e3$ belongs to category II.

Therefore, the objective for the sick probability formulation is not only to capture the anomaly from the golden mass, but also to weed out the operational state false positives.

We apply empirical Baysian estimation to derive the sick probability of a suspect entry: $P(sick) = \frac{N+c}{N+ct+cm(t-1)}$ where N is the number of samples, t is the number of suspects, c is the cardinality of the entry value, and m is the number of samples that matches the suspect value. This formula produces desirable results: Fixing all parameters except m, as m increases, the sick probability decreases; when $m = 0$, the derivative of the formula with respect to c is negative, indicating a decreasing trend of the sick probability as c increases. For detailed derivation and analysis of this formula, please see [5].

4. EVALUATION RESULT HIGHLIGHTS

We have prototyped a PeerPressure troubleshooter and evaluated its troubleshooting effectiveness over 20 real-world troubleshooting cases using a database that consists of registry snapshots from 87 machines. Registry size ranges from 77,517 to 333,193 entries, with a median of 198,608 entires. 87% of the registry entries have a cardinality of 1, 94% no more than 2, and 97% no more than 3.

The 20 troubleshooting cases for our experiments were all real-world failures that have troubled some users. And we have the knowledge of their root-cause misconfiguration a priori. Therefore, we use the ranking of the root-cause entry as our evaluation metric. To allow parameterized experiments, we reproduced these failures on a real-usage desktop using configuration user interface (e.g., Control Panel applets) to inject the failures whenever possible, and using direct editing of the Registry for the remaining cases. Then, we used “App Tracer” to get the suspects (see Figure 1). Finally, we ran PeerPressure to produce the ranking reports.

For the 20 cases, the number of suspects is large: ranging from 8 to 26,308, with a median of 1,171. Therefore, PeerPressure is an indispensible step of troubleshooting since sieving through these large suspect sets for root-cause entries is like finding a needle in a haystack. PeerPressure can effectively pinpoint the root-cause misconfigurations for 12 of the 20 cases. For the remaining ones, PeerPressure significantly narrows down the number of root-cause candidates by three orders of magnitude. For detailed presentation and analysis of the results, please see [5].

5. REFERENCES

[1] LARSSON, M., AND CRNKOVIC, I. Configuration Management for Component-based Systems. In *Proceedings of International Conference on Software Engineering (ICSE)* (May 2001).

[2] SOLOMON, D. A., AND RUSSINOVICH, M. *Inside Microsoft Windows 2000*, 3rd ed. Microsoft Press, September 2000.

[3] Web-to-Host: Reducing the Total Cost of Ownership, The Tolly Group.

[4] TRAUGOTT, S., AND HUDDLESTON, J. Bootstrapping an Infrastructure. In *Proceedings of LISA* (1998).

[5] WANG, H. J., CHEN, Y., PLATT, J., ZHANG, R., AND WANG, Y. M. PeerPressure, A Statistical Method towards Automatic Troubleshooting. Tech. Rep. MSR-TR-2003-80, Microsoft Research, Redmond, WA, Nov 2003.

[6] WANG, H. J., HU, Y.-C., YUAN, C., ZHANG, Z., AND MIN WANG, Y. Friends Troubleshooting Network, Towards Privacy-Preserving Automatic Troubleshooting. Tech. Rep. MSR-TR-2003-81, Microsoft Research, Redmond, WA, Nov 2003.

[7] WANG, Y. M., VERBOWSKI, C., DUNAGAN, J., CHEN, Y., WANG, H. J., YUAN, C., AND ZHANG, Z. STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support. In *Proceedings of LISA* (2003).