A *First* Approach to Understanding the Anomalies on My PC *

Luis M. Fernández, Hugo Terashima-Marín, Manuel Valenzuela-Rendón

{A00789695, terashima, valenzuela}@itesm.mx
Center for Intelligent Systems
Tecnológico de Monterrey, Monterrey Campus
Monterrey, Mexico

Abstract. Nowadays, and more than ever before, we live on a *computerized* world. Computers are used every day, all day, all around the world. All of people's activities now rely on PCs and, therefore, rely on the perfect execution of the applications running in these machines. As one can see, any malfunction of these machines may cause a huge problem in today's globalized world economy and safety.

This is a *first* approach to trying to understand such malfunctions (i.e., anomalies) that represent such a threat. The interaction of applications in the PCs are recorded and analyzed applying machine learning techniques, looking for possible combinations of applications that may lead to anomalies.

By applying simulation and machine learning techniques, some combinations of applications were found that, eventually, may lead to an anomalous event. Moreover, these combinations were clustered to see how similar they were, and concluding that there was a kind of defined set of combinations that may cause anomalies. Though, a simple approach is presented in this paper, the obtained results might be then used to improve the performance and profiling of everyday PCs. In this paper, all these findings are presented followed by the future work and conclusions related to this work.

1 Introduction

Computers have entered almost every arena of human society nowadays. They operate in our homes, our workplaces, and our schools. They come in so many shapes and sizes that it is sometimes difficult to recognize them: while laptop and desktop computers are commonplace, computers can also be found in home electronics, automobiles, airplanes, automatic teller machines (ATMs), security systems, and many other devices and situations. Many of the world's societies depend heavily on computers in the operation of their transportation systems, commerce, utilities, law enforcement, governance, and more [1].

From the above facts, it is easy to see that people's everyday activities rely not only on the applications that run on the PCs, but also on the correct and

* This project is funded by Tecnológico de Monterrey, Research Chair CAT 010

Advances in Artificial Intelligence and Applications
Research in Computer Science 32, 2007, pp. 193–203

Received 26/06/07
Accepted 31/08/07
Final version 20/09/07

flawless execution and interaction of these programs. Failing to appreciate the importance of the later may cause serious problems. Just to show this, consider the power grid's computers that keep power flowing in 15 USA states and one Canadian province. More than 50 million people were left without electricity on August 14, 2003, thanks in part to a software bug [2]. Moreover, consider the 13 Root Domain Name servers, two of which are operated by VeriSign, that each day receive 25 billion requests for Web addresses. This system is a frequent target of computer malcontents, and a February 6 attack caused three of the servers to drop 90% of their queries during a 12-hour period [2].

All these reasons, and many more [3,4], have created the need to develop tools and techniques to deal with these *anomalies*. Consequently, this has given birth to *Anomaly Detection* which can be described as an alarm for strange system behavior [5]. The concept stems from a paper fundamental to the field of security - "An Intrusion Detection Model", by Dorothy Denning [6]. In it, she describes building an *activity profile* of normal usage over an interval of time. Once in place, the profile is compared against real time events. Anything that deviates from the baseline, or the norm, is logged as anomalous.

Currently, there are many approaches that deal with anomaly detection. However, most of them look for anomalies analyzing network traffic or providing algorithms to detect intruders, without considering that anomalies are also present in just one PC (i.e., a home desktop). Some examples of current research are given by [7], [8] and [9]. In this paper, a first approach to detecting anomalies when using a computer is presented, looking at the interactions among typical applications. The main objective is to find common rules that might be useful to prevent the presence of anomalies. It is not possible that, given today's computer power, people are faced, occasionally, with a frozen application and/or total use of PC power by just one application.

In the following sections of this paper, the existing difference between this work and other publications is presented, the proposed approach to this problem is explained and detailed. Next, the implementation of this research is described, followed by the discussion and analysis of the obtained results. Finally, the conclusions of this current research are presented along with the future work needed to achieve the next stages of this research.

2 Related Work

Most of anomaly detection current research is devoted to the study of network traffic analysis and intrusion detection systems, profile generation and statistics and specification-based anomaly detection [6,7,8,9]. However, the present work is different in the sense that it attempts to study the complex interactions among applications that usually run in any PC. It is believed that before trying to study complex interaction among computers (e.g., distributed systems), it is important to understand what might go wrong or what might cause a PC and its programs to trigger a problem.

Anyway, there are some projects related to collecting data associated to the applications that run on a computer. On this, one of the latest projects is the Flight Data Recorder (FDR) [10] which logs all persistent state interactions of a PC. They propose a new way to store these logs and to access such information. However, the main objective of the data they provide is to improve not only how large systems are managed, but also how to prevent attacks (which they refer to as anomalies). As such, the present work is different since it attempts, first, to discover common application interactions that may lead to an anomalous event; and, second, to study how frequent such combinations appear when using a computer.

Jude Shavlik and Mark Shavlik [11] proposed the use of machine learning techniques to study combinations in PCs. Nevertheless, and even though the idea behind is the same, the way it is used and the goals are different. Jude and Mark Shavlik aim at finding intruders in the system whereas the present paper looks for the interactions of applications that may represent an anomaly.

This is an approach to studying the combinations of applications and the possible consequences that they may represent in the performance of any PC. The next section presents the steps taken to tackle this issue.

3 Approach

The followed approach is rather simple to explain. First, it is necessary to gather data, always the hardest thing to do. Each time step, all the processes that represent some PC load (not memory allocation) are recorded; these applications are usually those in use by the user. The applications in memory are not considered since they do not represent PC load, unless they are in use and not only loaded. Next, all this data is stored in a database for further analysis. In this analysis, rules that trigger anomalies are discovered and clustered employing machine learning techniques. In this case, three simple algorithms were tried out: k-means [1] [12], filtered clustering [2] [13] and farthest first [3] [14,15]. This approach is depicted in Figure 1.

However, there is still a problem that has to be overcome; there is a lack of available data to conduct our analysis. Although, getting data from a PC should be easy, there are some barriers that prevent doing so, being *privacy* issues the most important one. Almost no one is willing to let a program record all the programs a person makes use of [16]. This situation was solved simulating a PC and the programs running on it; however, there was the need of some statistics in order to get close-to-real-life results for this paper, more precisely:

- the most popular applications used by people, and

[1] The k-means algorithm is an algorithm to cluster objects based on attributes into k partitions.

[2] An algorithm that filters possible noise and unwanted attributes.

[3] A hierarchical clustering algorithm based on a measure of dissimilarity between observations.

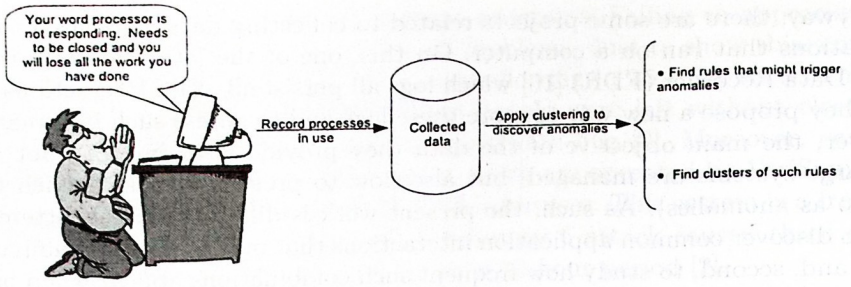

Fig. 1. Proposed approach to find anomalies.

– the percentage each of these applications represent as PC load

These data was provided by MugShot [17] where a complete list of most used applications plus their statistics and parent groups were provided. All these help simulate a PC running the most common applications and generate the required data to conduct the analysis.

Having overcome the lack of meaningful data, the next step is the implementation of this approach. This is explained and detailed in the next section of this paper.

4 Implementation

In this section the implementation of the proposed approach is presented. There are three aspects of the implementation that will be explained here: the simulation, the storage, and the machine learning tools applications.

4.1 Getting Data through Simulation

The simulation was conducted using Turtlekit [18], a MadKit plugin for multi-agent simulation. In this case, an abstract class was coded in Java, and all the applications, plus its characteristics, inherited from it. In the proposed simulation, each application is represented by an agent, instantiated with a different color, different life time, different PC load, and, for the purposes of the simulation, different position in the Turtlekit grid.

The different applications are instantiated at execution time using the statistics provided by MugShot web site [17]. Each application is associated with a probability, the applications that are used the most have a higher probability of being instantiated, whereas those that are seldom used have a smaller probability. Every new application that is placed in the Turtlekit world has its own characteristics (e.g., color, life time, etc.) This probability-instantiation approach is implemented using the genetic algorithm roulette wheel selection [19].

Figure 2 shows a moment of the simulation process that was conducted in order to get the data needed. Each little square represents a different application

Fig. 2. An example of the simulation process.

instance with a particular life time, so each application is in use for a period of time. Every time step a new application is instantiated and others *die*. Here is when the applications are recorded and stored for the next step and to do this another agent was coded. This agent looks at the grid and keeps a record of the agents alive at each stage of the simulation. This agent also keeps a sum of the percentages each in-use application represents. As long as this total sum is less than 100%, the possibility of having a *possible* anomaly is low. This is so under the assumption that most of the time computers do work well, otherwise, people would not make use of them [4]. On the other hand, once this total addition gets to 100%, then, a flag of *possible* anomaly is associated to the list of alive applications. The assumption that making use 100% of CPU resources could be a *possible* anomaly is not far from the truth. Whenever all CPU resources are being used there is a period of time when the PC simply does not respond. However, as the title suggests, this approach is a very first step towards really understanding what may cause an anomaly. There are other factors to be considered such memory paging, device usage, etc., which are not considered for the time being.

The process described above is conducted each step time, and the results are stored for further analysis. This storing process is described in the next section.

4.2 Storage

In this part of the project the applications in used are stored. These are stored as a word vector and a time marker. In other words, for each time t_i there will be a vector V_i that contains all the applications $[app_1^i, app_2^i, \cdots, app_n^i]$. The size of the vector is not constant since each time step there might be a different number of applications in use.

This last fact was the main reason why each group of applications was stored as a word vector. If these lists of applications had been stored as a key followed by the applications as possible attributes, an sparsed matrix would have been the result originating more problems than providing solutions.

[4] More on this assumption is discussed in the conclusions

4.3 Machine Learning Techniques Application

In this stage, machine learning techniques are applied to the data collected. It is important to point out that the data collected was quite huge. There were 10 runs of the simulation, where each run took 5 hours. The resulting size of the data sets after each simulation run was around 2GB.

It is clear to see that a very versatile tool was needed to conduct this stage of the research. As such, "Yet Another Learning Environment" (YALE) was used [20]. This platform provides a complete set of tools to conduct machine learning related experiments and analysis. Moreover, it is completely coded in Java which makes it easy to combine with other Java-based APIs; in this case MadKit.

Three machine learning algorithms were applied to the data [12,13] and [14,15]. All this algorithms considered the frequency of each single term present in each vector. In other words, these algorithms look for how many times an application app_i happens in each vector V_i. For the k-means algorithm [12], a random number seed of 10 was used and for farthest first [14,15], the random number seed was 1. The results gotten after using these techniques are presented in the next section.

5 Obtained Results and Discussion

All the data gathered after running the experiments was studied using machine learning algorithms. These results are presented in the next paragraphs. It is worth pointing out that in the following results, the proper names of the applications have not been included, and they will be referenced using their main use.

5.1 Mined Rules

Using the algorithms already mentioned, a set of rules was mined trying to characterize what applications may trigger a possible anomaly. In this case, only those lists of applications that were flagged as *possible* anomalies were studied.

The results gotten are shown in Table 1. It is interesting to see that the presentations utility is present in two of the four rules. Also, it can be seen that the email manager is also present in two of the mined rules. Also something quite interesting to see is that just the music player application might be enough to cause an anomaly.

Table 1. Mined rules that may trigger a possible anomaly.

Rule1:	presentations utility	and	email manager
Rule2:	word processor	and	presentations utility
Rule3:	email manager	and	document viewer
Rule4:	music player		
Total Number of Mined Rules:			4

Table 1 tells us that there are certain applications, and a combination of these, could eventually lead to an anomaly when using a PC. However, it is also important to point out that this results were gotten using a simulation process.

5.2 Formed Clusters

The word vectors were clustered according to how similar they are. In other words, how many applications they share and how many they do not. The three clustering algorithms that were used show quite similar results. Nevertheless, in one of them the number of members of one cluster is different from the number of members in others. The number of clusters in three algorithms does not change, though.

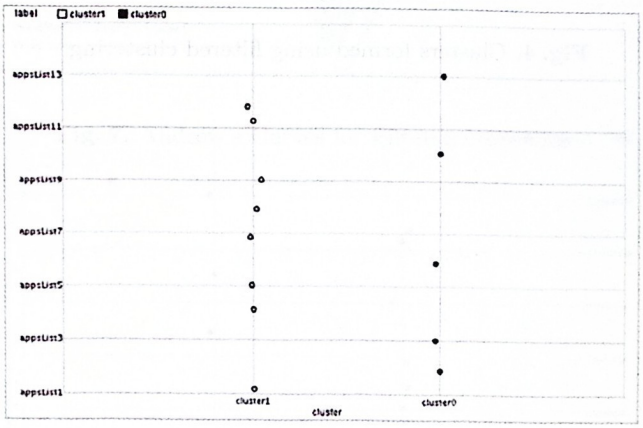

Fig. 3. Clusters formed using K-means.

Figures 3 and 4 show the obtained clusters using k-means and filtered clustering respectively. It is interesting to see that both have the same pattern and distribution on the grid.

On the other hand, Figure 5 shows the clusters using the farthest first algorithm. In this case there are two vector words that are similar to those shown in Figures 3 and 4. All these clusters just confirm that the applications in those word vectors might, indeed, lead to an anomaly when using a PC.

Furthermore, it is also important to show how *similar* this word vectors might be. In order to do so, Andrew's Curves [5] were plotted using the data provided thanks to the implementation of [21]. Figures 6, 7 and 8 show such plots.

As expected, Figures 6 and 7 show almost similar behavior and, they also show how coherent the clustering is. Figure 8 is different from the other two; however, the coherence that this graph shows is also good validating the clusters.

[5] Andrew's Curves are a useful tool for separating multivariate observation into groups that can not easily be distinguished in a tabular presentation, where curves of similar observations generally overlap, while dissimilar observations fall into different groups.

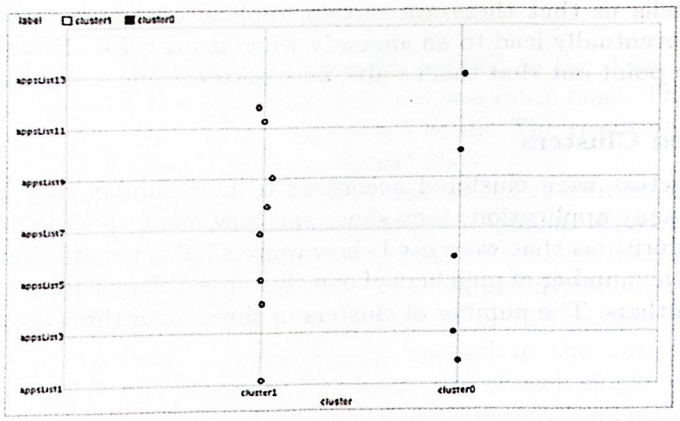

Fig. 4. Clusters formed using filtered clustering.

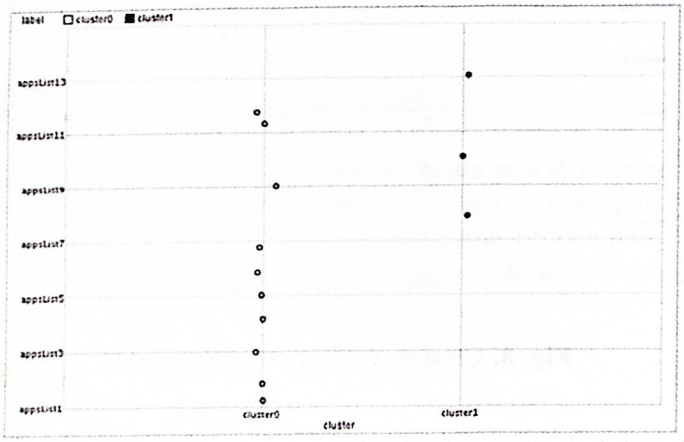

Fig. 5. Results using farthest first clustering algorithm.

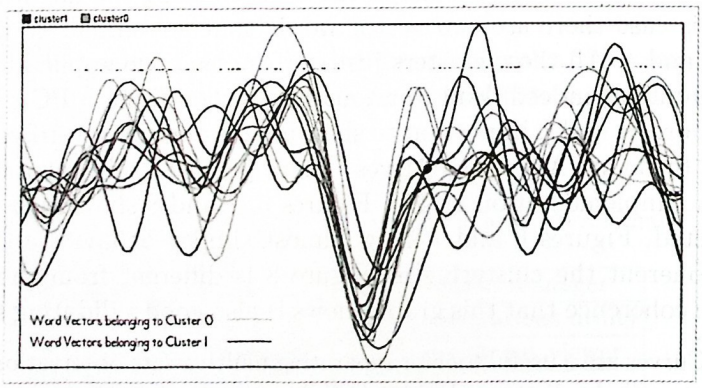

Fig. 6. Andrew's curves for K-means.

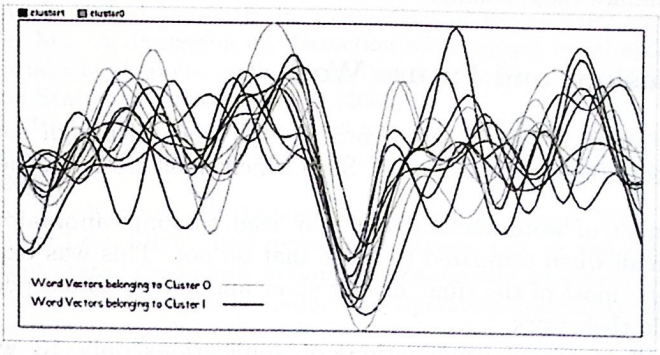

Fig. 7. Andrew's Curves for Filtered clustering.

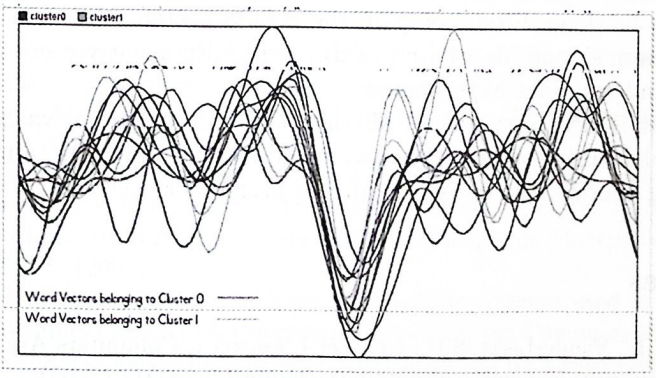

Fig. 8. Andrew's Curves for Farthest First.

All these results show that the rules found might be warnings that indicate possible anomalies and that the sets of applications that potentially lead to anomalous events do share some common characteristics. In the next section, the conclusions of the present work are reported along with the future work needed to validate these results.

6 Conclusions and Future Work

The present research has obtained some results that, although some of them were expected, others were not at all. Such conclusions are listed below.

1. The number of word vectors that may lead to some anomalous events is rather small when compared to those that do not. This was expected since computers, most of the time, do not show anomalies. Thus, people rely on PCs to do their work.
2. Among all generated combinations of applications only 13 word vectors caused an anomaly flag.
3. The applications that could potentially lead to an anomaly is also very small. The rules that were built show that only 5 of all 21 most common applications represent some threat.
4. In general, the possible combination of applications that may lead to an anomaly is the same across all seen flagged word vectors. This was shown when they clustered in a rather tight way.

Nevertheless, there are still some things left to be done, and the most important one is to go beyond the *simulation* and actually get real data. There are some approaches to such task [10]; however, they are not totally specific to the situation under studied here. Once this is done there are two other things that have to be done. First, compare the results gotten with real data versus simulation; and, second, find false positives and false negatives crossreferencing simulation and real data experiments.

Finally, it is necessary to use all this new knowledge to warn of *possible* anomalies when using a PC. One way could be to implement an entity that monitors the applications in use, building a profile per each user.

References

1. Nyhoff, J.L., VanderLeest, S.H.: Chapter 1, Lesson 1: Computers Are Everywhere. www.calvin.edu/academic/rit/webBook/chapter1/lesson1/ (2005)
2. Hesseldahl, A.: Our Fragile Computerized World. (2007)
3. Reeves, G.E., Neilson, T.C.: The Mars Rover Spirit FLASH anomaly. In: Proceedings of the Aerospace Conference, IEEE Computer Society Press (2005)
4. Associated-Press: Engineers labor to fix space computers. www.msnbc.msn.com/id/19224133 (2007)

5. Tanase, M.: One of these things is not like the others: The state of anomaly detection. http://www.securityfocus.com/infocus/1600 (2002)
6. Denning, D.: An Intrusion-Detection Model. In: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1986)
7. Deri, L., Suin, S., Maselli, G.: Design and implementation of an anomaly detection system: An empirical approach. In: Proceedings of Terena TNC. (2003)
8. Basseville, M.: A discussion on 'Detection of intrusions in information systems by sequential change-point methods' by Tartakovsky, Rozovskii, Blažek, and Kim. Journal on Statistical Methodology **3** (2006)
9. Kim, S.S., Reddy, A.L.N.: Image-based anomaly detection technique: Algorithm, implementation and effectiveness. Journal on Selected Areas in Communications **24**(10) (2006)
10. Verbowski, C., Kiciman, E., Daniels, B., Kumar, A., Wang, Y.M., Roussev, R., Lu, S., JuhanLee: Flight data recorder: Always-on tracing and scalable analysis of persistent state interactions to improve systems and security management. In: Proceedings of the Seventh Symposium on Operating Systems Design and Implementation (OSDI). (2006) 117 – 130
11. Shavlik, J., Shavlik, M.: Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. (2004) 276–285
12. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: The analysis of a simple k-means clustering algorithm. In: Symposium on Computational Geometry. (2000) 100–109
13. Witten, I., Frank, E., Trigg, L., Hall, M., Holmes, G., Cunningham, S.: Weka: Practical machine learning tools and techniques with java implementations. In: Proceedings of ICONIP/ANZIIS/ANNES'99 Int. Workshop: Emerging Knowledge Engineering and Connectionist-Based Information Systems. (1999) 192–196
14. Hochbaum, Shmoys: A best possible heuristic for the k-center problem. Mathematics of Operations Research **10**(2) (1985) 180–184
15. Dasgupta, S.: Performance guarantees for hierarchical clustering. In: 15th Annual Conference on Computational Learning Theory. (2002) 351–363
16. Web-Page: Privacy and Security (BS7799, ISO 17799, Computer Forensic). www.netlitigation.com/netlitigation/privacy.htm (2007)
17. Web-Page: Application Statistics - MugShot. http://mugshot.org/applications-learnmore (2007)
18. Michell, F.: Introduction to turtlekit: A platform for building logo based multi-agent simulations with madkit. Technical Report RR LIRMM 02215, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Université Montpellier II (2002)
19. Blickle, T., Thiele, L.: A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich (1995)
20. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: Yale: Rapid prototyping for complex data mining tasks. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06), Springer (2006)
21. García-Osorio, C., Maudes, J., Fyfe, C.: Using andrews curves for clustering and sub-clustering self-organizing maps. In: Proceedings of the European Symposium on Artificial Neural Networks - ESANN2004. (2004) 477 – 482