USER ACTIVITY DETECTION IN COMPUTER SYSTEMS BY MEANS OF RECURRENCE PLOT ANALYSIS

Tomasz Rybak[1], Romuald Mosdorf[1]

[1]Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: As computers are getting faster and disks are growing bigger more data describing user behaviour can be gathered. These data can be analysed to gain insight into user behaviour and then to detect user traits. Currently many different methods are used to analyse data — and there is still no one best method for analysing different parameters of computer systems. Computer systems behave non-linearly because they run many programs on multi-user operating systems; this causes inter-program dependencies requiring non-linear methods to analyse gathered data.
The aim of the article is to present how non-linear methods are able to detect subtle changes introduced into system by user's actions. Number of interrupts per second was chosen as variable describing system's behaviour. Analysis presented in this article focuses on idle system and system busy accessing hardware. Article shows that using recurrence plot can reveal similarities in behaviour of the system running different programs, and therefore can be used to detect similarities and differences in users behaviour.
This article presents analysis of system activity through usage of series of recurrence plots to detect changes introduced by user actions. Analysis of lengths of horizontal and vertical lines on recurrence plots allows for describing periodicity of the system. This allows for gaining insight into behaviour of entire computing environment. Article shows that different tasks (such as network transmission, writing or reading from CD-ROM, compressing data) result in different recurrence plots; at the same time changes introduced by those tasks are hard to detect without analysis data. This means that usage of recurrence plot is crucial in detecting changes introduced in system by user's actions.

Keywords: user behaviour analysis, fractal analysis, recurrence plot

1. Introduction

Computers are typically driven by their users. Even if a computer is running programs that were not chosen directly by user, this is done to aid user: anti-virus program

Zeszyty Naukowe Politechniki Białostockiej. Informatyka, vol. 5, pp. 67-85, 2010.

protects against harmful software, firewall reduces risk of attacks from the network, indexing system scans all documents to allow for faster finding interesting phrases in the files. Programs mentioned above run all the time, so their characteristics can be treated as background noise while analysing the system's behaviour. Most of the characteristics of a running system come from programs that were directly run by the user. Some users tend to run many programs at the same time and switch between them; others perform tasks inside one application, close it when job inside this program is done, and only then start a new program. Some noise can be introduced by the fact that some programs use other programs — e.g. text processor can call spreadsheet or graphics program to compute results and generate graphs. Even in the case of a single user, her behaviour can depend on the mood, eagerness to work, degree of tiredness, time of day, etc. Therefore, the view of entire system, and change of it over time, can be used to characterise behaviour of the user.

As noted by Rolia et al. [7], knowing state of system, their capacities, possible bottlenecks, and current load allows for reconfiguration to avoid unnecessary overloading of some machines by routing load to other ones. Porter and Katz [5] are measuring details of behaviour of systems to have roughly the same load on all servers so none is over- or under-utilised. Matthew Garret [1] notes that to be able to reduce power consumption, and thus ecological footprint of computers, we need to know details of behaviour of individual programs and entire system. Most common ones are: how many operations involving disk are executed in one second (pointing whether the disk can slow down), how often CPU needs to check state of peripheral devices (disallowing switching to the lower power consumption modes), and whether any operations can be grouped together so there are longer gaps between operations allowing for disabling of some hardware.

Hoffman [2] and Rogers [6] describe monitoring of server farms serving Hotmail mail and Microsoft pages respectively. They note that to be able to manage large groups of machines and to be able to detect anomalies one needs to gather details of their work. On the other hand, no human can cope with such amounts of data so one needs to use statistics to detect trends and base decisions on those aggregate results. Hoffman also notes that it is impossible and pointless to try to come with artificial test cases when managing large groups of machines. Generating large tests requires large amounts of work and thinking about many different scenarios. To be able to execute test cases one also needs need fair amount of servers — which would be used solely for testing and not for usage by users. The biggest disadvantage of using dedicated testing machines would be inability to come with all of scenarios that users can generate. There is too many users, possible software configurations, and so on. He claims that it is better to just observe and analyse behaviour of real, life system. This requires ability of accessing necessary data and very fast responses to any problems. The most important in such a case is disallowing for analysing software not to cause performance cost on the systems it is running on.

Oskin [4] claims that increasing number of cores in CPU forces programmers to analyse data describing execution details so he is able to check if multi-threaded programs behave correctly and to correctly manage large number of virtual processors. George Neville-Neil [3] observers that current operating systems provide programmer with access to many internal hardware counters that can show how well program is executing and if its performance suffers on particular hardware platform. Shaw et al. [9] use specialised hardware with good amount of monitoring to make sure that massive parallelism is well used and no chip is using power without doing useful work. Counters in such systems include cache hits and misses, number of context switches, number of correctly and incorrectly predicted branches, page faults, etc. Although they are most useful for operating system programmers, they point why computer is behaving slowly and thus are valuable resource of information about internals of computer system behaviour. But again they generate vast amounts of data which is very hard to analyse by human being.

As can be seen from cited literature there is still no consensus over which methods are the best and which ones in the best way represents behaviour of programs, especially when different aspect of behaviour are analysed. This article presents usage of non-linear methods to detect characteristics of user's behaviour. Analysed activities were using hardware present in the computer system: transmission over network, burning files to CD and reading files from CD, so number of hardware interrupts per second was used as the best descriptor of the system behaviour. Interrupt usually occurs as result of hardware event, like keyboard or mouse activity, disk or network transmission, comes from hardware clock, etc. The more interrupts, the more activity comes from hardware, which means that programs are intensively communicating with the outside environment.

Because of inter-program dependencies we assume that computer systems are non-linear dynamic systems. The aim of the article is to present how non-linear methods are able to detect subtle changes introduced into computer system by user actions. The remainder of the article is structured as follows. The next section describes procedure used to collect data, including hardware and software configuration. Section 3. describes theory of mathematical means used to analyse gathered data. Section 4. describes and analyses obtained results and their meaning. The two last sections summarize paper and present possible future research.

2. Methodology of data collection

Data was gathered on the AMD Duron 1.3GHz with 768MB or RAM and single IDE 7200RPM hard drive. Debian Linux system (version named Sid) was used as operating system for this computer. System had 1GB of active swap partition; kernel was 32-bit 2.6.26 with Debian patches. System was not upgraded (neither manually nor automatically) during entire course of experiment to avoid changes in the environment that could influence process of gathering data or the data itself.

System had public IP address, so it was possible to connect to it from the Internet. Network connection was not disabled, because doing so would not resemble normal mode of operation that would be later examined. Some run-levels, however, have not started any servers listening on the network which allowed for comparison of situations with and without public services available. We claim that having active network card does not invalidate experiment results.

Data was gathered by running the computer on different run-levels (set of running system processes) on consecutive days. To collect numerical data vmstat program was used. Detailed description of characteristics of Unix-like operating systems, as well as details of process of collecting data can be found in our previous paper [8].

As mentioned in Introduction, number of interrupts per second was chosen as a measure of system's activity. Data was gathered using vmstat program once a second for about 90 minutes, depending on the observed run-level. Data from all possible run-levels was gathered. We decided to use four cases: one from a system in which only a bare necessity of programs needed to run the operating system is run (Single mode, raw data in Figure 3 a)), one from a system in which all programs except graphical environment are running (level 2, raw data on Figure 4 a)) one from a system with active graphical environment (level 5, raw data on Figure 5 a)) and finally from a system in which user is logged in, and he was transferring data over network, copying it, burning to CD drive and reading data from CD (raw data on Figure 6 a)).

3. Non-linear signal analysis

Recurrence plot was chosen to analyse the gathered data describing characteristics of the Linux system.

Fractal analysis of an one-dimensional signal assumes that all important dynamic variables present in system influence these time series. To perform non-linear (fractal) analysis of the signal we need to transform this signal into one describing point in a high-dimensional phase space. To do it we treat few consecutive values as coordinates of one point in the phase space. Usually all values are normalised at the very beginning of analysis to simplify computations. Number of values taken from a stream and treated as coordinates of points depend on the dimensionality of phase space. Usage of all points that were captured in such a way results in attractor reconstruction. In many cases not all values are used — this is called "stroboscopic coordinates". To avoid visual clutter only one of every N points is drawn. Number of non-drawn points is determined by parameter τ, called time delay; it is multiply of time between points of original time series. Dimension and Lapunov coordinates of original attractor and attractor reconstructed using stroboscopic coordinates and using all points from original signal are the same.

At the same time choosing proper value of time delay τ influences our ability to analyse the signal. If τ is too large input points lie too far away from each another to provide enough information. If it is too small, input points lie too close which may suggest presence of non-existent linear dependency in the signal. One of the possible methods to find proper value of time delay is to find period (even non-exact one) and to choose value slightly smaller than found period. If there is no visible period, one can use autocorrelation (and take half of maximum value) or mutual information. In case of mutual information we take value of the first minimum of this function.

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{1}$$

This function is constant or oscillates while τ increases when data is generated by periodical systems. In chaotic system value of this function decreases rapidly when τ increases. This function is therefore useful for determining whether underlying mechanism is periodic or chaotic.

Recurrence plot is based on idea of calculating attractors. It is used to compare all possible states that are represented by trajectories of points in the high-dimensional phase space. If such trajectory goes through region that is close to previous one, it is matched as recurrent. Recurrence plot is the chart showing all periods when dynamic system's state is repeating. Usually phase space has dimension much larger than possible to visualise and understand by human. Recurrence plot (proposed by Eckmann in 1987) allows to show on 2D chart all repeating states of system and is based on matrix of similarity. Positive Lapunov points are matched by diagonal lines' lengths.

Recurrence plot can be defined as Haeviside function over difference of distance of points is space (over some metrics) and the threshold. Its main three parameters are τ, dimension and threshold ε. Too small threshold means that some points that are far away will be taken as close ones; such situation can occur in the system where much of the values are very small and from time to time there is large spike, like ECG signal.

$$R_{ij} = \Theta(\varepsilon - \|x_i - x_j\|) \tag{2}$$

Single recurrence plot is 2D matrix of values from set of $\{0,1\}$. Recurrence plot is symmetrical amongst diagonal. It can be plotted on the screen or the paper. Black dot (value of 1) at coordinates (i, j) means that on system at time i and j was in similar state, because its attractor was represented as points that were close together (their distance was less than the chosen threshold). This means that dot is plotted if two sequences coming from input data are similar (their product is larger than threshold). This allows for visually analysing similarity of signal at different scales. Similar techniques are used in analysis of gene sequences (FASTA, BLAST) to find similar gene sequences. This technique requires large amounts of memory and long processing.

Recurrence plot can be used as a mean for visual analysis of the self-similarity of signal, but also to find numerical characteristics of analysed dynamic system. The most important parameter used in analysis described in this paper is laminarity showing how stable the system is. This can be determined by measuring length of horizontal (or vertical, as recurrence plot is the symmetrical matrix) lines. According to the web page http://recurrence-plot.tk/ horizontal (and vertical) lines point to the periods where system does not change much. Another important factor is divergence, pointed by length of diagonal lines. Diagonal lines point to the states where system is oscillating and trajectory returns to the close subspace. It can be connected to positive Lapunov exponents and point where signal is repeating itself.

Dividing entire signal into parts and generating recurrence plots for each of them results in series of recurrence plots. This allows for temporal analysis of lengths of lines present in the plot. Technique used in this article creates many recurrence plots, each starting one point later than the previous one. For each of calculated plots lengths of horizontal and diagonal lines are calculated; then maximum lengths of appropriate lines create charts used in analysis of system (and user) behaviour.

4. Characteristics of gathered data

Figure 1 shows the first 25 components of FFT signal. Single mode (shown in part b) of the figure) has the lowest values of power, hence it is presented on a separate chart. Its values are about 500 times smaller than those for other situations. At the same time this chart shows the largest variation of frequencies present in signal. This

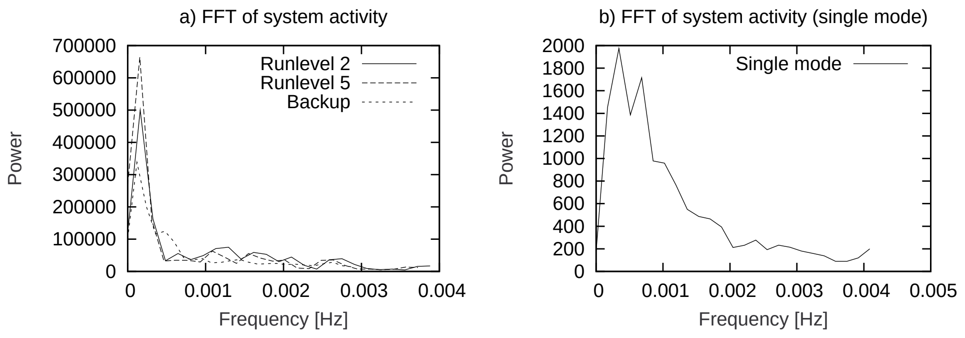

Fig. 1. Power spectrum of analysed signal. a) Run-level 2, Run-level 5, and user activity; b) Run-level Single

means that in case of Single run-level noise is much more visible. This is similar to the so-called pink noise.

Other run-levels' frequencies are in similar range, although they differ in exact values of power present in signal. All of those signals (shown in part a) of Figure 1) have very distinctive main frequency. This was caused by disk activity during scanning disk in search of suspicious files. In the case of the last data set, presenting system with running user-initiated programs, it generates second frequency in the chart. The large first part of tail in the case of backup is caused by inter-connection of programs that were being run during backup process. Usage of the results from one program by another one can cause resonance visible in the figure.

Figure 2 shows interdependence of the original signal and signal after some time. The least amount of mutual information in the signal in in the Single mode, which.is to be expected, as the smallest number of programs were running there. The amount of mutual information in signal from Single mode does not decrease over time. Run-levels 2 and 5 contain more information. They have similar overall shapes of curves but chart of run-level 2 is more smooth. This comes from fact that in this mode less programs are running (no digital clock, no screen-saver, no other graphical utilities), so there is less interaction. But similar shape means that changes caused by interactions between those new programs do not have long-lasting consequences. This suggests that those programs do not run for long time and also one program do not cause effects that affect running of other programs. Signals coming from run-level 2 and 5 show small decrease in value of mutual information. At the same time mutual information in the case of user interaction with the system is much larger, and decreases very rapidly.

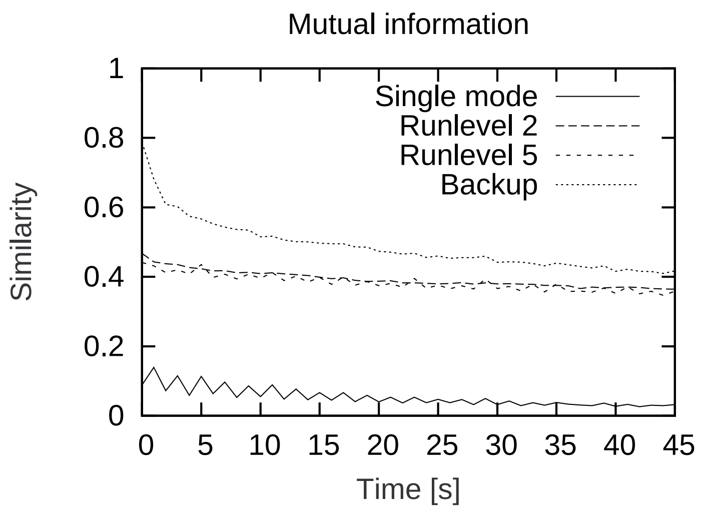

Fig. 2. Mutual information of signal compared to itself after time

Recurrence plots was created with following values of parameters: $\tau = 3$, dimension $m = 2$ and threshold $\varepsilon = 0.02$. Detailed analysis of single recurrence plots was described in [8]. This article is extension of previous work.

Window of size 150s was chosen for analysing series of recurrence plots. Analysis for windows of size 60, 90, 120, 150, 180, and 240s was performed initially. For short windows images were too noisy, and longer windows resulted in disappearance of fine details. Hence our decision of size of window 150s. Of course for different situations (especially different types of collected data) it might be necessary to chose different size of window.

N-150 recurrence plots were generated for signal of length N. Then, for each of recurrence plots from the window the longest diagonal and horizontal lines were found. Following charts show how much those maximum lengths were changing over time of experiment.

Figure 3 a) shows shows number of interrupts per second in single mode, in which almost no service was active; Figures 3 b), c), and d) show sample recurrence plots that were generated from this data. Although mean activity was constant, one can see changes in the signal. In the beginning (first 3 minutes) signal shows great variability. This is visible on the recurrence plot Figure 3 b) and is caused by post-startup activity of the system. This is rather early phase of running of the system

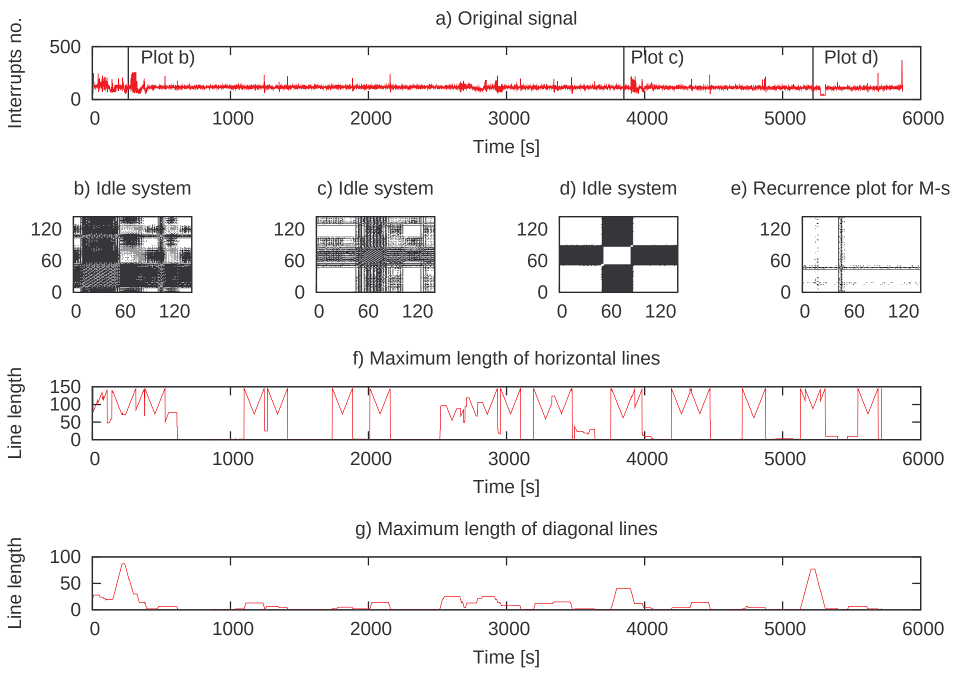

Fig. 3. Recurrence Plot for Single run-level. a) Original signal; b)-e) recurrence plots; f) maximum lengths of horizontal lines g) maximum lengths of vertical lines

which means that some programs are still initialising itself. In later chart signal is much more nice. Only part shown in Figure 3 c) shows increased activity; because, as mentioned in Section 2. system was connected to the internet, it is probable that it was someone trying to connect to the system.

Figure 3 f) shows maximum length of horizontal lines As mentioned in section describing recurrence plots (Section 3.), horizontal (and vertical) lines point to stationary parts of the system, when system is laminar (it does not change or changes very slowly). One can see that there are periods when horizontal lines are large, and when they are short. In most situations we can observe "M-shaped" structures. At the beginning there is no horizontal line, then there is jump to maximum length, then some slight but steady decrease of length, but not much (less than 50%) and again growth, and then almost immediate drop to zero. Those occurrences are connected to the spikes in the signal, and they can be used to detect such rapid changes in values in signal.

Diagonal likes (Figure 3 g)) are not long, as system is not changing much, and long diagonal lines point times when system is rapidly changing.

Figure 4 shows number of interrupts in run-level 2, without active graphical environment. Figure 4 b) shows recurrence plot from the period of disk-scan, and part c) situation when there was not activity. Situation seen in the latter figure is not interesting, and is similar to situation from run-level Single. This plot differs from the one shown in [8] as it does not show any details. This is due to choosing threshold which meant that for the idle period no points were printed. On the other hand choosing lower threshold would mean that recurrence plot in the period of activity would be almost entirely black; values differ over ten times between those cases. Changing threshold in course of analysis is something to investigate but it could taint data and results. Normalisation of data can also influence shape of recurrence plot. We are not yet sure how to deal with comparing raw recurrence plots from highly variable signals.

Figure 4 b) shows fragment of run-level 2 during high activity (disk scan, Phase 1). Signal changes but some self-similarity can be seen. Figure 4 c) is also run-level 2, but this is fragment after disk scanning, in the idle mode (Phase 2). Here recurrence plot is almost entirely black: this means that there is much similarity in signal. This also means that there is no long-lasting changes in signal, at least not enough to be detected by the plot. This situation differs from from the Single run-level. Additional daemons (programs constantly running in the background) caused change in activity and system switches more often to serve them, which results in more variable signal.

Analysis of idle period did not result in any points in recurrence plots. The only non-zero parts in graphs of maximum lengths of horizontal and diagonal lines,

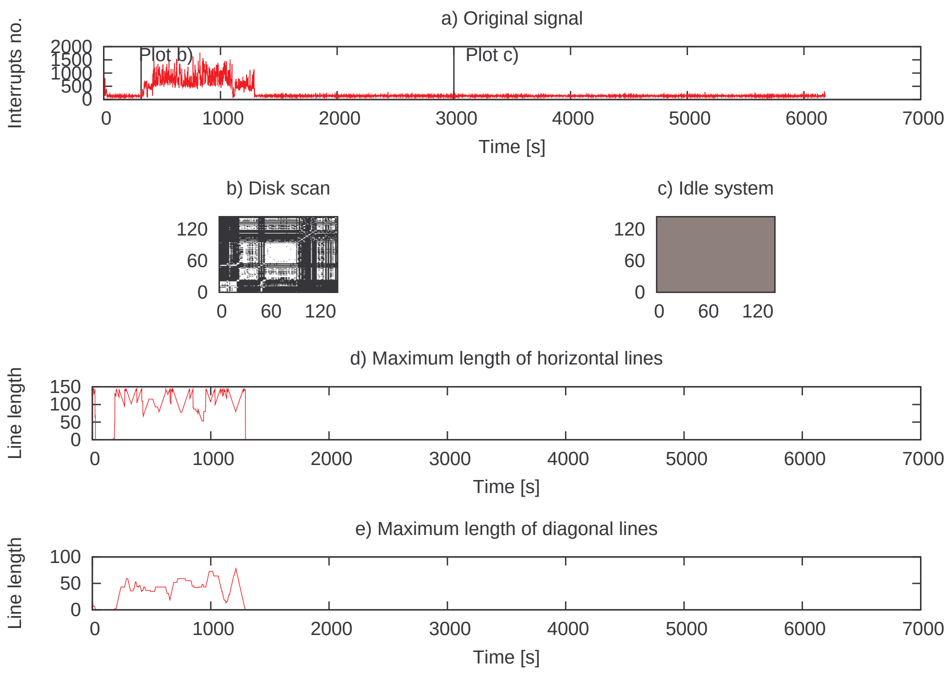

Fig. 4. Recurrence Plot for Run-level 2. a) Original signal; b) and c) recurrence plots; d) maximum lengths of horizontal lines e) maximum lengths of vertical lines

showing stability (Figure 4 d)) and divergence (Figure 4 c)) of system are those from the full disk scan.

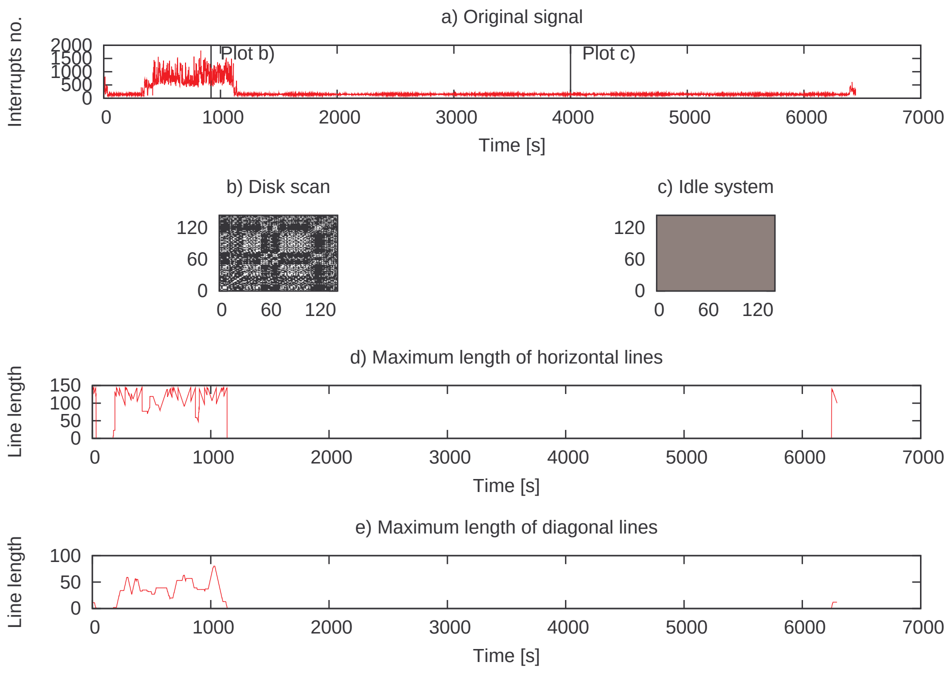

Fig. 5. Recurrence Plot for Run-level 5. a) Original signal; b) and c) recurrence plots; d) maximum lengths of horizontal lines e) maximum lengths of vertical lines

Figure 5 a) presents number of interrupts in run-level 5 with active graphical session. Here situation is very similar to situation in run-level 2, described in previous paragraphs. Similarity of those two situation was also described in previous paper [8], Both recurrence plots and graphs of stability (Figure 5 d)) and divergence (Figure 5 e)) look almost exactly the same.

For run-level 5 Figure 5 b) shows recurrence plot during disk scanning (Phase 1). This plot is similar to one seen in run-level 2 (part a of this figure). But in case of later activity, after disk scanning (Figure 5 c), image is different from run-level 2. This means that although there is self-similarity in signal, it is not on so many levels. Here again additional graphical programs cause “ripples’ that cause signal to be less smooth. The same situation was seen in information plot (Figure 2) and in FFT (Figure 1).

Figure 6 a) shows active graphical environment with user logged-in and during backup creation. We can see four different phases: transmission of files over the network, shown in Figure 6 b), compression of all files and making them ready to save to the CD (Figure 6 c)), burning data to CD (Figure 6 d)), and reading files from CD (Figure 6 e)).

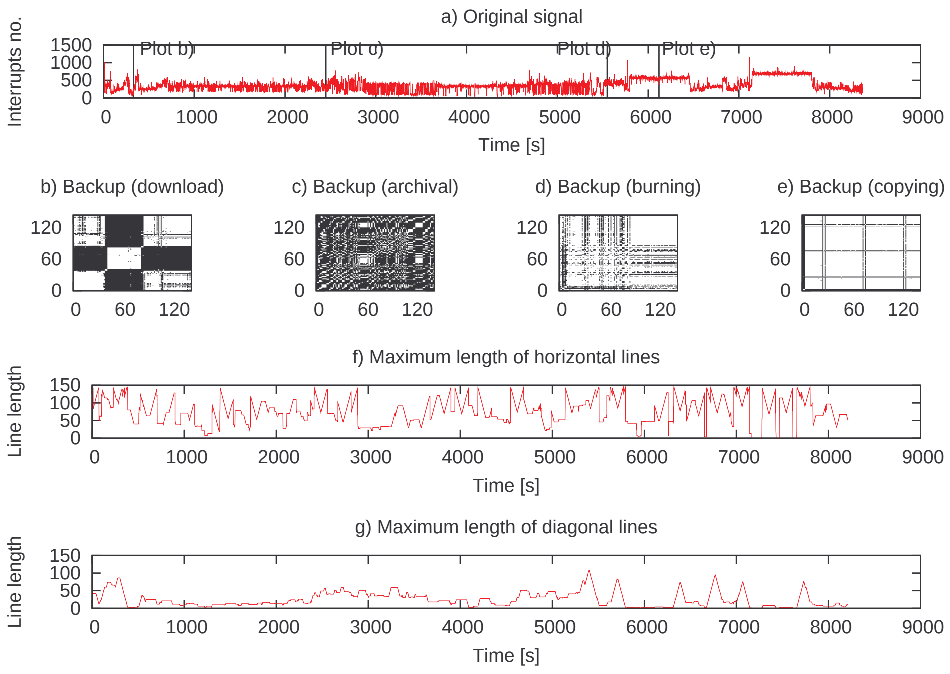

Fig. 6. Recurrence Plot for system with user activity. a) Original signal; b)-e) recurrence plots; f) maximum lengths of horizontal lines g) maximum lengths of vertical lines

5. Discussion

Figure 2 shows signal's mutual information. It shows that introducing user's behaviour changes amount of inter-dependency of signal with respect to time. Charts showing idle systems are flat; adding new programs (system daemons in case of run-level 2 and graphical environment for run-level 5) increases values but does not change shape of the system's mutual information. When user is active, however, situation changes dramatically. Mutual information is much larger and drops rapidly after the first few seconds. This means that user's activities affects both long- and short-term memory of the system, but as the time passes traces of user's behaviour disappear and inter-dependency of signal presenting user's activity decreases to levels present in systems without user's actions.

This is the result of running programs working on the same task (creation of backup copy of entire system) so programs depend on each other as results generated by one program is input data for the next one. Running more programs causes introduction of information into the system. Further analysis of mutual information is needed to determine influence of different tasks: CPU-bound, limited by memory, disk, and other subsystems on mutual information charts.

First three figures showing recurrence plots (3, 4, and 5) show situations without any user activity thus showing idle system state. One can observe that first chart presents rather constant mean activity with small variation. Next two charts show two different stages; one (shown in Figures 4 c) and 5 c)) is similar to single run-level but with slightly larger mean value. Both of them show large activity about two minutes after system start and lasting for 10 to 15 minutes. This is caused by service monitoring changes in files on entire hard-drive checking every file on disk. This lasts and also caused running of many processes.

From analysis of system's configuration (details in [8]) it is clear that single mode (Figure 3) can be treated as entry (basic) level and presents the least sophisticated situation. Its activity changes only slightly, and can be treated as constant. This activity comes mostly from kernel responding to hardware and time events. Run-levels 2 and 5 are quite similar, but 5 has higher level of base (average) activity (when no user programs are running) because more programs are running in the background as run-level 5 activates graphical environment, as opposed to run-level 2. They both present much activity starting at 5 minutes from system start up to about 20 minutes. This is caused by cron job that is checking validity and consistency of installed packages. After finishing it no other non-standard program is running.

As expected, running more processes in the higher run-levels introduces more variability to signal. On the other hand when entire system is busy running the same task situation is similar regardless of active run-level, as can be seen on Figures 4 a) and b) and 5 a) and b), which are very similar — even if the former comes from system without any X-Window programs, and latter from active X session. But in this situation entire system is busy checking all files present on hard drive — so any differences disappear eclipsed by this activity.

As seen in Figure 6 e) showing self-similarity of signal during burning image to CD, hardware-induced events are very regular. This means that system is influenced not only by running programs and user's actions but also by external sources of events, like network (where arrival of packet causes kernel to serve it) or hardware events (sector read from the disk, empty buffer in the CD-ROM burner, sample ready to be read from the sound card). As system has no control over external environment and only partial control over hardware used to communicate with physical environment, those events can be seen as coming from outside of the system.

Figure 6 c) presents recurrence plot of CPU-intensive task during backup procedure (compression of files). We can see that there is high level of self-similarity. Figures 6 d) and e) show signal during disk operations (burning image to CD). This signal has very distinctive structure with period of about 40 seconds. It is caused by buffering: kernel transmit data to DVD drive until its buffers are full, and then can do switch to other tasks. When drive has no data it informs kernel which then again transmit data needed to fill-up buffers. We can estimate that drive has buffer sufficient to store data for about 40s of work.

Figure 6 g) shows behaviour of system through lengths of diagonal lines. At the very beginning length of diagonal lines grows which means that system is repetitive. After the end of disk scan maximum length of diagonal lines dropps significantly which means that system's behaviour is more chaotic. After starting backup procedure the length of diagonal lines started growing; it means that repetitive process was running again. During creation of backup maximum lengths of diagonal lines was changing: it decreased, increased, then some spikes started to show. Those changes point presence of some processes that were reoccurring at very few occasions.

Figures 4 d) and 5 d) show maximum lengths of horizontal lines in recurrence plots for run-levels 2 and 5. In those run-levels for the first twenty minutes was busy checking all files on the hard drive, and then went idle waiting for user interaction. The final minute of activity should be not taken into consideration as it shows activity during system shutdown, as described in section 2.. In both of those cases system is showing stationary and recurrent behaviours only during disk-scanning phase. Idle system does not show any recurrent or stationary behaviour which means that idle system presents chaotic behaviour. Programs run by user can thus be seen as behaviours that introduce order into initially chaotic system. This is confirmed by Figure 6 f) which shows increased lengths of horizontal lines. It means that running programs increase amount of stability in the system.

"M-shapes" appear again on Figure 6 f); similar shapes were present on Figure 3 f). They are again caused by spikes present in the original activity chart. Figure 3 e) shows horizontal lines that span through almost entire chart. Horizontal (and vertical) lines do not come through entire plot but are separated at the crossing. There is small gap in the lines where they cross; it is not visible in the chart with the naked eye, but it exists and limits length of the line in the chart. This means that when the crossing is in the middle of the plot the line is the shortest. The line gets longer when lines move away from the middle, as more of the line is not separated. The horizontal line is thus the longest when gap is at the border of chart. When this gap moves to the center of the plot, length of horizontal line decreases. This is the first part of the "M-shape"; then gap moves away from the chart, length of line grows and second part the this shape begins. The exact source of system's activity causing occurrence of spikes in the signal is currently unknown. Because the main purpose of presented research was detection of changes caused by user's actions, network was not disabled to limit changes introduced to system by process of conducting experiment. This could be caused by some network activity, like ARP requests, tries of infecting machine, etc. Those "M-shapes" on chart showing maximum length of horizontal lines can be used to detect such rapid changes in signal. This behaviour is not visible on other run-levels (shown in Figures 4 b) and 5 b)), though. Horizontal lines on the system with user's requested actions are much less regular, as can be seen in Figure 6 b). This additional source of events introduced by user's actions is more powerful and can occlude subtle differences introduced by external network packets coming to the system.

Lengths of diagonal lines in Single run-level recurrence plot is smaller than for other run levels (Figure 3 c)). Charts showing maximum lengths of diagonal lines in run-levels 2 and 5 (Figures 4 c) and 5 c) respectively) look similar to charts showing horizontal lines from those recurrence plots (Figures 4 b) and 5 b)).

6. Summary

This article presents analysis of system activity using series of recurrence plots and lengths of lines present on those plots. Number of interrupts fired in each second was measured and treated as a signal describing the system activity. Data from four different configurations of the system was gathered, with and without user activity. We have shown that system activity can be investigated and explained using non-linear methods.

Signal was analysed using a mutual information and recurrence plots. We have shown that introducing user to the system increases amount of information present in the system and the lone value of mutual information can be used to determine whether the system is idle or if user is active. Shape of mutual information plot can be used to determine whether the system is running short-living programs, or whether it is busy with long-lasting, intensive tasks.

The majority of our investigation focused on using of recurrence plots as means of analysing changes in the signal and detecting trends. Human user is operating on time scale of minutes so gathering signal once every second and analysing trends over many seconds is a good compromise between detecting events in the system and limiting amount of data to analyse.

Research described in this paper focused on using a number of interrupts as a measure of the system activity. Number of interrupts is an important variable in cases with small sets of running programs and when running programs are dependent on hardware (network transmission, reading and writing CD). Context switch means stopping one program and starting another. It is used in modern operating systems to achieve multiprocessing, or at least illusion that many programs are running at the same time. The larger number of context switches, the more frequently kernel switches tasks. A large number of contest switches means that there are many programs running on the system, and all of them are waiting for CPU (i.e. not many are waiting for some external event). Number of context switches, as noted in Section 2., can be useful in analysing systems in which many programs are running. A difference between those variables and their interdependencies call for further research.

As noted in Section 3. many parameters can influence shape of recurrence plot. Further research is needed to investigate influence of parameters (like threshold ε, or time delay τ) on results that can be obtained during analysis of data. Differences between charts presented in previous ([8]) and current article show that it is crucial to use proper values of parameters. Improper values can cause disappearance of important details in the noise. Automatic procedure of generating proper values of parameters can be very helpful with analysis of user's behaviour.

Like Hoffman [2] we claim that observing real system is crucial; this is why we use existing tools that do not influence system much. We also intend to find ways of real-time automatic analysis of gathered signal. As noted in a comment to CACM article[1]: "Another [usage of growing computing capabilities] might be defensive computer security, analyzing past and current patterns of activity on the machine, communicating with other machines, and working to prevent malicious activity."

We were able to show that analysis of computer system activity allows to detect changes introduced by user's behaviour. User's actions change state of the system through running programs and requiring processing of data. This changes state of the system; but those changes are not always visible in the signal without performing analysis on it. Presented analysis using recurrence plots generates chart which allow easily to detect user's behaviour, such as transferring data over the network, writing

[1] http://cacm.acm.org/blogs/blog-cacm/23833-what-to-do-with-those-idle-cores/fulltext

data to compact disk, compressing data, or reading data from the external storage. By analysing characteristics of the recurrence plots and periodicity of the data we were able to detect rapid changes in the signal and distinguish between different signal types, which leads to detecting user activity that generated particular signal shape.

Acknowledgements

We would like to thank the anonymous reviewer for thoughtful and valuable comments that improved the article, and to our colleagues that helped with the language mistakes.

References

[1] Matthew Garrett. Powering down. *Communications of ACM*, 51(9):42–46, 2008.

[2] Bill Hoffman. Monitoring, at your service. *Queue*, 3(10):34–43, 2005.

[3] George V. Neville-Neil. Kode vicious beautiful code exists, if you know where to look. *Communications of ACM*, 51(7):23–25, 2008.

[4] Mark Oskin. The revolution inside the box. *Communications of ACM*, 51(7):70–78, 2008.

[5] George Porter and Randy H. Katz. Effective web service load balancing through statistical monitoring. *Communications of ACM*, 49(3):48–54, 2006.

[6] Daniel Rogers. Lessons from the floor. *Queue*, 3(10):26–32, 2005.

[7] Jerry Rolia, Ludmila Cherkasova, Martin Arlitt, and Vijay Machiraju. Supporting application quality of service in shared resource pools. *Communications of ACM*, 49(3):55–60, 2006.

[8] Tomasz Rybak and Romuald Mosdorf. Computer users activity analysis using recurrence plot. In *International Conference on Biometrics and Kansei Engineering*, Cieszyn, Poland, 2009. AGH.

[9] David E. Shaw, Martin M. Deneroff, Ron O. Dror, Jeffrey S. Kuskin, Richard H. Larson, John K. Salmon, Cliff Young, Brannon Batson, Kevin J. Bowers, Jack C. Chao, Michael P. Eastwood, Joseph Gagliardo, J. P. Grossman, C. Richard Ho, Douglas J. Ierardi, István Kolossváry, John L. Klepeis, Timothy Layman, Christine McLeavey, Mark A. Moraes, Rolf Mueller, Edward C. Priest, Yibing Shan, Jochen Spengler, Michael Theobald, Brian Towles, and Stanley C. Wang. Anton, a special-purpose machine for molecular dynamics simulation. *Communications of ACM*, 51(7):91–97, 2008.

WYKRYWANIE AKTYWNOŚCI UŻYTKOWNIKA PRZY UŻYCIU ANALIZY RECURRENCE PLOT

Strzeszczenie: Dzięki nieustannemu wzrostowi wydajności systemów komputerowych możemy gromadzić coraz więcej danych opisujących aktywność systemów. Dane te mogą być analizowane aby zyskać wgląd w zachowanie użytkowników. Uważamy że systemy komputerowe, z racji działania w nich wielu programów które wpływają wzajemnie na siebie, majaą charakter nieliniowy. Dlatego też spośród wielu istniejących metod analizy dużych zbiorów danych zdecydowaliśmy się na użycie nieliniowych metod analizy.
Artykuł przedstawia wykorzystanie nieliniowych metod w celu wykrycia subtelnych zmian wprowadzonych do systemu poprzez działanie użytkownika. Analiza skupia się na porównaniu systemu bezczynnego i takiego w który działają programy uruchomione przez użytkownika. Jako zmienna najlepiej charakteryzująca system została wybrana liczba przerwań na sekundę. Artykuł przedstawia użycie wykresu recurrence plot w celu wykrycia podobieństw w zachowaniu systemu, a przez to w działaniu użytkownika.
Badanie systemu wykorzystuje serię wykresów aby wykryć charakter zmian wprowadzonych przez użytkownika. Analiza długości pionowych i ukośnych linii pozwala na wykrycie okresowych zachowań komputera, a tym samym na lepsze zrozumienie procesów zachodzących w całym systemie. Pokazane zostało że różne zadania (transmisja danych przy użyciu sieci komputerowej, nagrywanie plików na dysk CD, odczyt plików z dysku DVD, kompresja danych) generują różne wykresy recurrence plot. Ponieważ zmiany stanu systemu nie znajdują odzwierciedlenia w sygnale przedstawiajaącym liczbę przerwań na sekundę, użycie recurrence plot jest kluczowe do wykrycia zmian spowodowanych przez użytkownika.

Słowa kluczowe: badanie aktywności, aktywność systemu, analiza fraktalna, recurrence plot

Artykuł zrealizowano w ramach pracy badawczej S/WI/2/09.