Computer users activity analysis using recurrence plot

Tomasz Rybak, Romuald Mosdorf Computer Science Departament
Bialystok Technical University (BTU)
Bialystok, Poland
Email: (rybak, mosdorf)@wi.pb.edu.pl

ABSTRACT

We can currently observe trend of gathering vast amounts of data about systems and users behaviour. Gathered data is analysed to get insight about behaviour of users and then to detect traits of users. Currently many different methods are used to analyse data and there is still no one best method for analysing different parameters of computer systems. We use non-linear methods to analyse gathered data because we assume that computer system is the non-linear one because of inter-program dependencies in current multi-program, multi-user operating systems.

To best describe behaviour of entire system we choose number of interrupts per second as analysed variable because this value shows overall state of activity of programs being run on the system. We show that using recurrence plot can show some similarities in behaviour of system, and therefore can be used to detect similarities and differences in users behaviour.

I. INTRODUCTION

As noted by Rolia et al. [6] 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 [4] use similar approach to have roughly the same load on all servers so none is over- or under-utilised.

Hoffman [3] and Rogers [5] describe monitoring of server farms serving Hotmail mail and Microsoft pages respectively. They note that to be able to manage large amounts of machines and to be able to detect anomalies one needs to gather details of their work. On the other hand one cannot cope with such amounts of data and needs to use statistics to observe trends and base decisions on those aggregate results. They claims that it is better to just observe and analyse behaviour of real, life system instead of dedicating subset of machines for testing.

As described work shows, trend of gathering vast amounts of data about systems behaviour can be observed. At the same time attempts of limiting space of results are visible — human mind is not very good with managing many simultaneous variables and getting their interdependencies. Also, many existing methods of detecting trends, anomalies, etc. are best suited for scalar values, not vectors consisting of many variables. In some cases we are not fully interested in absolute values but rather in trends, and having more than few graphs is discouraging and makes vary hard to choose appropriate parameters.

The same data is also useful for distinguishing one user from another. Each user behaves differently — even if two users are given the same tasks, they will do it in different manners. One, for example, will run all programs at the same time and will try to do his tasks concurrently, while other will try to do them sequentially. As will be shown later, those modes of operations can give different results, which can be then used to detect traits of users.

We assume that computer system is the non-linear one because of inter-program dependencies in current multi-program, multi-user operating systems. We use non-linear methods to analyse gathered data. The aim of our paper is testing non-linear methods for analysing data describing entire computer system and individual programs' behaviour.

The aim of the paper is to present how well non-linear methods show internal state of system at the same time vastly decreasing amount of data observer is faced with. It is first stage of our investigation of non-linear analysis of dynamics of computer programs working in observed environment therefore we chose Linux system as our platform and its following parameters. We intend to

gather more data describing behaviour of different programs and investigate it further.

This paper is divided into three parts: first describes system and programs used to gathering data (including different configuration modes of Linux system), methodology of observation and gathered data set. Second describes non-linear analysis methods used in this work and results of their usage on gathered data. Third is presentation of results and future research plans.

II. Linux working characteristics

The foundation of Unix system is small kernel and set of processes that serve user. Because those systems are used in many environments and their usage patterns vary with time there is need to alter set of programs that are being run. This is done by concept of run-levels. Each run-level describes set of programs that are started and stopped when entering into this particular level. Conventions of run-levels in Linux-based systems are described in literature: their concept and management in document called "Linux Standard Base" ([2])[1] which intention is to describe standards that different distributions adhere to so programs can be run on any Linux system, and in "Filesystem Hierarchy Standard" ([1])[2] , describing placement of all files in Linux system.

Run-levels are managed by init program; it is run as first process after kernel startup. All scripts that can be used during system startup or shutdown are placed in set of special directories, described in chapter 3.7 of [1]. They describe which programs to run, in which order, and whether service should be started or stopped. Management of those scripts (their format, creation, enabling and disabling) is described in Chapter 20.4 of [2]. Run-level described by letter S is called "Single"; this mode is usually used for trouble shooting. It only starts system, without starting any additional services. Logging any users except administrator is not possible. Run-level number 2 is normal mode for text environment, with all daemons running and waiting for connections. This is default run-level when no graphical environment is installed — however some systems do not differentiate between level 2 and 3 and treat them the same. Run-level number 5 is used for starting graphical environment Besides that it is similar to run-level 2 and 3.

[1]Documents are available free of charge at http://www.linuxfoundation.org/en/Specifications or http://refspecs.freestandards.org/lsb.shtml

[2]Available at http://www.pathname.com/fhs/

Details of those run levels are described in Part VII (System Initialization), Chapter 20.5 (Run Levels) of [2][3]

III. Methodology of data collection

Data was gathered using AMD Duron 1.3GHz with 768MB or RAM and single IDE 7200RPM hard drive. Operating system used to gather data was Debian unstable. It is systems used on daily basis by one of authors and therefore details of its work and configuration are known. This allows for easier hypothesis creation and changes of configuration required to test them. System has 1GB of active swap partition; kernel is 32-bit 2.6.26 with Debian patches.

Systems was not upgraded (neither manually nor automatically) during entire course of experiment to avoid changes in environment that could change data.

Program called vmstat was used as primary source of data. It is standard program present in many Unix-like systems; in standard behaviour it calculates and presents on the screen statistics about workings of entire system. It can take as parameter how often (how many seconds apart) it should gather data and present new results. Default value (used in course of described experiment) is one second. This is also the smallest interval that can be set in this program.

Number of pages taken by swap, free, treated as cache and buffers for dirty files were counted. Also number of blocks read and written to disk and to swap space, percent of CPU time spend on user processes, systems, waiting for IO completion and idle. Each of those values was gathered with 1s accuracy.

Program vmstat when run with default configuration records 16 values that are grouped in 6 sets. Two first values describe processes: how many wait for CPU and disk time. Second group consists of 4 values describing memory, including amount of RAM used as disk buffers, free, etc. Third group consists of two values describing swap behaviour and values from this group can show whether system lacks memory. Fourth group consists of two variables describing behaviour of IO subsystem: number of blocks read from devices and written to various devices per second.

Fifth group describe activity of processes in system. First value (in) presents number of interrupts

[3]Available at http://refspecs.linux-foundation.org/LSB_3.2.0/LSB-Core-generic/LSB-Core-generic/runlevels.html

per second including clock ones; interrupt usually occurs as result of hardware event, like keyboard or mouse activity, disk or network transmission, comes from hardware clock, etc. This is value that we investigate in our article. Second value (cs) is number of context switches per second; context switch is stopping one program and starting another, to achieve multiprocessing, or at least illusion that many programs are running on the same time. The larger this value, the more frequently kernel switches tasks.

The last group consists of four values that describe in which states CPU (processor) spends time: percentage of CPU time spend in user and system space, amount of time CPU is idle and waiting for input/output operations. Those four values should add to 100% but sometimes, because of rounding errors, difference of 1 can be observed. This means that in rare circumstances sum can be 101 or 99%.

A. Gathered data description

Configuration of init program was changed so it asks for desired run-level at system start. This allowed for avoiding running all programs from default run-level and later stopping them, as those operations would alter data and greatly increasing difficulty of correct analysis.

Script starting vmstat which later would write gathered data to log file was written. It was placed in directory /etc/init.d/ together with other scripts run at each system start by init program. For each run level system was running without human intervention (except run-level 1) for about 90 minutes, and later root user would log in, and run reboot program to restart system.

Data describing run-level Single, run-level 1 and run-level 4 were gathered during single day, in exactly that order. Data sets describing other run-levels were gathered on consecutive days. Spreading data gathering for many days was necessary because all of them run cron program which default configuration is to run set of programs daily. Running two runlevels on the same day could mean that one of them executes daily chores and other does not — which could mean that gathered data describe different situations. Run-levels 1 and Single do not run cron daemon, so they can be tested on the same day.

Because vmstat was set to write results once a second and was run for about 90 minutes, about 5400 instances of 16-value vector was gathered for each of tests (here different run-levels).

Interrupts number per second was chosen for further analysis, as it presents the largest variability

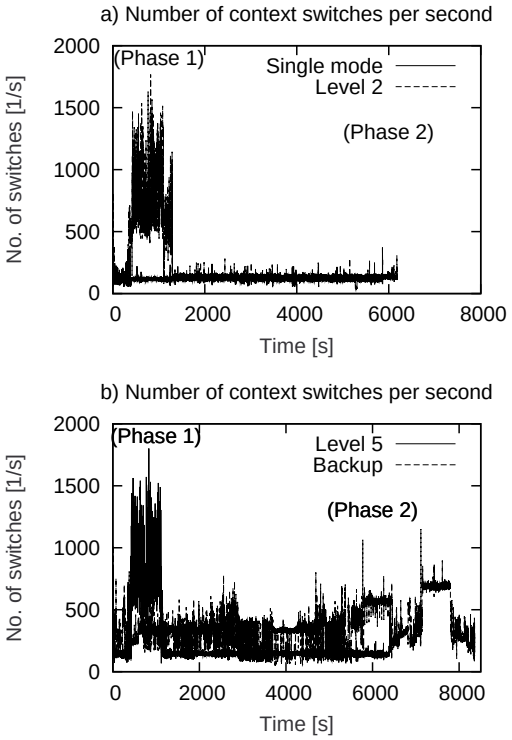

Fig. 1. Number of context switches per second in all modes

in entire experiment. In our opinion other variables are also interested but this value shows overall state of activity of programs being run on the system. The largest value of this variable, the more system is active.

Figure 1 shows number of interrupts (value described by symbol in) in four different situations. First is single mode, in which almost no service was active; second is from run-level 2, without active graphical environment; those are shown in part a) of figure. Third is run-level 5 with active graphical session, and last is active graphical environment with user logged-in and during backup creation — shown in part b) of Figure 1. On those both graphs one can see two interesting periods, described here by “Phase 1” and “Phase 2”. Former in case of run-level 2 and run-level 5 was when cron job which was checking all files on hard drive was running. In case of creation of backup this phase was compression of files, using large quantities of CPU power and time. Phase 2 was in case of levels 2 and 5 when system was in idle state — nothing extraordinary was running, only programs that always are present. In case of backup during that second phase CD image was physically written (burnt) into disk.

First three situations were without any user activity thus showing idle system state. One can observe that first presents rather constant mean activity, but with some variation. Next two show two different stages; one is similar to single run-level but with slightly larger mean value. But both of them show large activity about two minutes after system start and lasting for about 10 to 15 minutes. This is caused by service monitoring changes in files on entire hard-drive which needs to check every file on disk. This lasts and also caused running of many processes.

As can be seen on Figure 1 those four situations differ. Single mode 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) – this comes form fact that more programs are running in the background as in run-level 5 graphical is running, unlike in 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.

The last situation presents large activity during entire process of creating copy of system. It consisted of downloading some data from other machines, copying and compressing local files, crating DVD image, burning it into physical disk and verification of created medium. Some operations were IO-bound (copying, burning), and some were CPU-bound (compression, verification).

IV. Non-linear analysis using recurrence plot

From many different non-linear methods for analysing data we decided to use recurrence plot as basis of our investigation.

Recurrence plot is shows self-similarity of signal at different time scales. It is calculated by using two-dimensional array. Point at X and Y means that signals starting at time X and time Y were similar to each another. This allows for seeing if signal shows self-similarity, which means that it is repeating itself. 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. However this technique requires large amounts of memory and long processing.

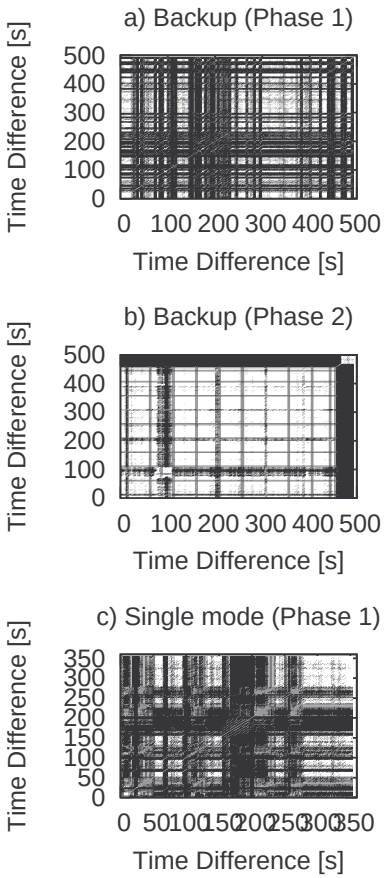

Fig. 2. Recurrence plot for single mode and backup

Because this technique for signal length N draws matrix NxN and uses $O(N^3)$ time we were not able to run calculations for entire signals — we did not have enough memory to finish calculations.

Figure 2 c) shows recurrence plot for Single mode. Some similarity visible in original signal (Figure 1 part a) is confirmed by bottom chart (recurrence plot itself). There are similar fragments, except for middle where some event caused signal to change. But this new signal is also self-similar which can be seen on the bottom plot, but with less time scale.

Figure 3 a) shows fragment of run-level 2 during high activity (disk scan, Phase 1). Signal changes but some self-similarity can be seen. Figure 3 b) is also run-level 2, but this is fragment after disk scanning, in 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 in the plot.

For run-level 5 Figure 3 c) 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 3 d), image is not so black. This means that although there is self-similarity in signal, it is not on so many levels. Here again additional graphical programs cause small changes visible in signal.

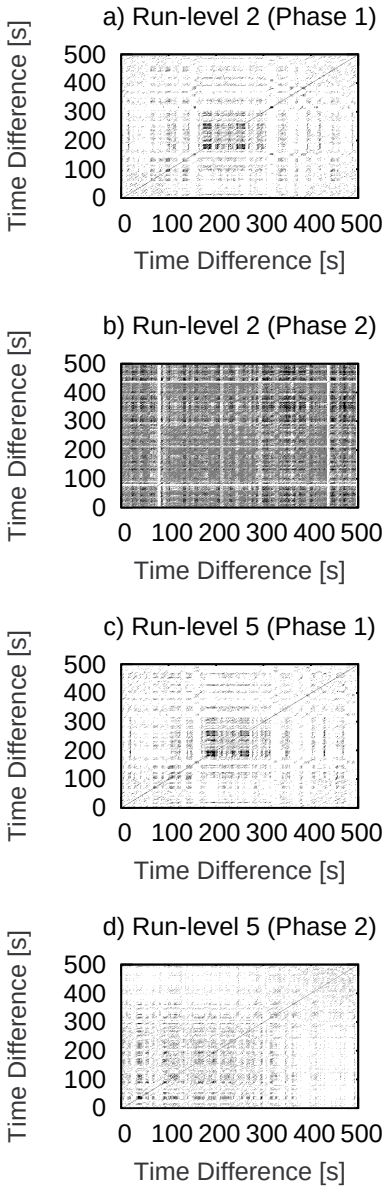

Fig. 3. Level 2 and 5 recurrence plots

Figure 2 a) presents recurrence plot of backup procedure. It is fragment that was during high CPU activity (compressing files, Phase 1 of Figure 1 b). We can see that there is high level of self-similarity. Figure 2 b) shows signal during disk operations (burning image to DVD, Phase 2). We can see that it is rather interesting: signal has very distinctive structure with period of about 40 seconds. This comes from buffering: kernel transmit data to DVD drive up it is full, and then can resume previously done tasks. When drive has no data it informs kernel which then again transmit data needed to fill-up buffers. So we can estimate that drive has buffer sufficient to store data for about 40s of work.

V. Results and discussion

Like Hoffman [3] we claim that observing real system is crucial. We use existing tools and our methods give some interesting results. We believe that we have shown that system activity can be investigated and explained by using non-linear methods. Recurrence plots show that gathered data (when treated as signal) shows signs of self-similarity.

As expected, running more processes (going to higher run-levels) introduces more variability to signal. On the other hand when entire system is busy running similar tasks situation is similar regardless of active run-level, as can be seen on Figures 3 a) and c). They are very similar even if the former comes from system without any X-Window programs, and latter from active X session. Any subtle differences disappear during this activity. Situation looks entirely different when system returns to idle state, as shown in Figures 3 b) and d). They show the same run-levels (2nd and 5th, respectively) after all maintenance jobs are done and system is not busy Each of active programs runs with different speed and has different periods of activity which means that entire system is less regular, so different sets of active programs will show different characteristics.

Activity of system can be influenced by external source of events as can be seen in Figure 2 b) showing self-similarity of signal during burning image to CD. Here we can see that hardware-induced events are very regular. On the other hand events induced by user influence also system, but are much less regular.

It should be noted that this is beginning of our work. From description of vmstat tool reader can see that there is more data to be analysed. We believe that also analysis of inter-values is needed to fully understand system and users behaviour.

References

[1] Filesystem hierarchy standard 2.3. Technical report, Filesystem Hierarchy Standard Group, jan 2004.

[2] Linux standard base core specification 3.2. Technical report, Linux Foundation, jan 2008.
[3] Bill Hoffman. Monitoring, at your service. *Queue*, 3(10):34–43, 2005.
[4] George Porter and Randy H. Katz. Effective web service load balancing through statistical monitoring. *Communications of ACM*, 49(3):48–54, 2006.
[5] Daniel Rogers. Lessons from the floor. *Queue*, 3(10):26–32, 2005.
[6] 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.

This work was supported by the grant S/WI/2/2008 from the Bialystok Technical University.