UNAME Tool: Automatic Generation of Computer Resources Monitoring Scripts

Jean Araujo*‡, Carlos Melo†, and Cézar Cordeiro*
*Universidade Federal do Agreste de Pernambuco, Garanhuns, Brazil
†Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil
‡Universidade Federal de Sergipe, São Cristóvão, Brazil
jean.teixeira@ufape.edu.br, casm3@cin.ufpe.br, cezar.cordeiro@dcx.ufpb.br

***Abstract*—Decision-making regarding performance issues requires computer systems monitoring at any level of service provisioning. However, most of the available tools do not always allow flexibility, or when it does, it impacts directly the resource consumption through an intrusive methodology. This paper presents the UNAME tool, which proposes the automatic generation of Shell scripts for monitoring computational resources. Through a GUI, the user can drag in or out the components to be evaluated. The Linux monitoring commands were mapped to high-level representations of the computational resources, and now the user, through little interaction or knowledge about scripting, can generate their own scripts to monitor the required system resources. To demonstrate the UNAME tool feasibility we provide a case study and an evaluation scenario. The results demonstrated the effectiveness of the proposed tool.**

***Keywords*—Monitoring, Evaluation, Shell Scripting.**

I. Introduction

Countless times, before making important decisions for projects and actions, users or system administrators need to check the current status of computational resources at their disposal. Such verification is required to ensure that the decision made does not generate losses, such as misallocation of resources, system unavailability, or even financial losses [1]. Therefore, it is an necessary to observe and evaluate computer systems.

There are several ways to perform computational resources monitoring. The non-exhaustive list goes from commercial software, free software, or even scripts written in several languages that may include some native command lines arguments. Linux and Windows are two of the mainstream operating systems used in both clients and server-side; both of them provide a way to check their resources consumption [2]. However, the use of Linux-based systems by service providers is at another level. Almost 96%[1] of all web servers run Linux-based operating systems.

Shell scripts have been widely used in several studies related to the performance evaluation field, such as for monitoring Android applications [3], cloud platforms [4]–[6], hypervisors [7], and so many others. This paper presents the UNAME tool, a front-end that can help general users, administrators, and researchers to monitor computational resources using automatically generated scripts in a simple and faster way. The proposed tool focused on monitoring Linux-based systems through Shell scripting language. The main advantage of adopting this technique is the possibility to perform non-intrusive monitoring, which means that the monitor have lower impact over the general performance of the monitored system [8].

The remainder of this paper is organized as follows. Section II presents the basics concepts about performance evaluation and Shell scripting. While Section III provides an overview of the UNAME tool, and Section IV shows a case study and an evaluation scenario that demonstrates the tool feasibility. Already Section V presents some related tools and compares them with the UNAME tool. Finally, Section VI presents the conclusions and future works.

[1]https://www.zdnet.com/article/can-the-internet-exist-without-linux/

II. Background

This section describes the fundamental concepts for understanding this paper, what includes subjects on Performance Evaluation and Shell Scripting.

A. Performance Evaluation

Performance evaluation deals with the process of measuring and analyzing the performance of a particular system under test (SUT) [9]. Usually, it aims at the system's understanding and documentation based on specific situations, as well as on the administrator needs, interests and skills [1].

Usually, most of the performance evaluation focuses only on the time that takes for the system to perform a given task, but the reality is much bigger than this. We often analyze some specific parts of the whole in order to obtain values about other characteristics than time itself. If it is interesting to analyze the network performance in terms of bandwidth, regardless of memory or processor, then the performance evaluation should be carried on aiming this issue. But setting a threshold is a really difficult task, and demands from the system analyst a well-known basis about what the system does and how it does, only this way he/she can establish beforehand what are the metrics and components to be monitored.

To achieve most of performance evaluation goals, creativity is a needed gift that can be acquired through time, and is used to select the techniques and to monitor the system without disturbing it, or keeping the external impact as little

as possible, while providing accurate results. After data collection, the analyst can use statistical techniques to interpret the results [1] and provide to decision-makers ways to reach some conclusions about the system.

B. Shell Scripting

Shell scripts are byte-sized programs used to automate general tasks performed either by common users or systems administrators [10]. Most operating systems have a built-in Shell environment that can be accessed through a command-line interface. We may define Shell itself as a set of commands that can be used to check some information about a system, hardware, and software component. While the Shell scripting corresponds to adding these commands to a general file with some extension (e.g *.sh*) that can be later executed and obey structural and sequential logic.

As can be seen in the script below, line #1 presents the interpreter's instruction or the *shebang* line. Then, in line #2, the header is added. The echo command is used to display strings on standard output. With the operator >> it is possible to add a line to the current end of an output file (log.txt). The while loop in line #3 is intended to record in the report the output result of the commands in line #4. In this case, the monitoring will only stop when the user finishes executing the script. The instruction in line #5 is the monitoring frequency, that is, before starting the next iteration of the while loop the algorithm "sleeps" for 10 seconds [2].

Algorithm 1: General Monitoring pseudo-code

```
1 #/bin/bash
2 echo "Var1 Var2" >> "log.txt"
3 while true do
4 |   echo $(mon-Var1) $(mon-Var2) >> "log.txt"
5 |   sleep 10 ;
6 end
```

III. UNAME Tool: An Overview

The UNAME tool was developed for the web to guarantee multi-platform portability. The tool was designed as a single-page application (SPA), which is a web application or website that interacts with the user by dynamically rewriting the current web page with new data without the need to load entire new pages [11]. The main goal of this approach is to provide faster transitions that make the tool looks like a native app. In this sense, the tool was designed so that all actions were taken in the same place. The tool has a main screen, where it is divided into 4 main sections. Figure 1 presents the system's main screen.

The entities to be monitored by the UNAME tool were called nodes. A node is a virtual instance of the component that will be monitored, and each node has its specific configuration. Five types of nodes were defined: Computer, CPU, Network, Disk and Processes. For each node, there are a set of resources to be monitored. Table I shows the resources for each node.

To correctly perform its activities, the UNAME tool requires some built-in components, as well as the installation of some auxiliary prerequisites. Also, it is necessary to have a basic knowledge to run Shell scripts on the Linux terminal [12]. Table II shows the aforementioned commands.

TABLE II
Requirements for running scripts

Requirements	Commands
Built-in	date, mpstat, free, df, cat, ps aux.
Installable	echo, awk, cut, gawk, grep, head, sleep, wc.

A. Features

The menu provides the four main features of the UNAME tool, which are: (i) export template; (ii) import template; (iii) run button; and clear. The **export template** feature provides a mechanism to export monitoring schemas to a JSON file. This feature consists of a modal window that presents to the user two different interactions, the first is a fast copy of the JSON, while the second one is the export button, enabling the model's JSON to be saved on the users computer, as can be seen in Figure 2.

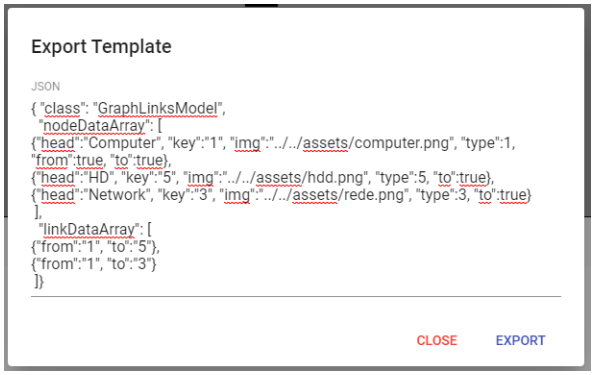

Figure 2. Template's Export

As expected, the **import template** feature provides a way to restore a previous exported template. This feature is also presented in a modal window, with a text field for importing the template, as can be seen in Figure 3. Once the user presses the import button, the system moves itself to an architectural modeling area.

Figure 3. Template's Import

The third feature of the menu is the **general settings**. Figure 4 presents it, which is divided into three steps. The first is the

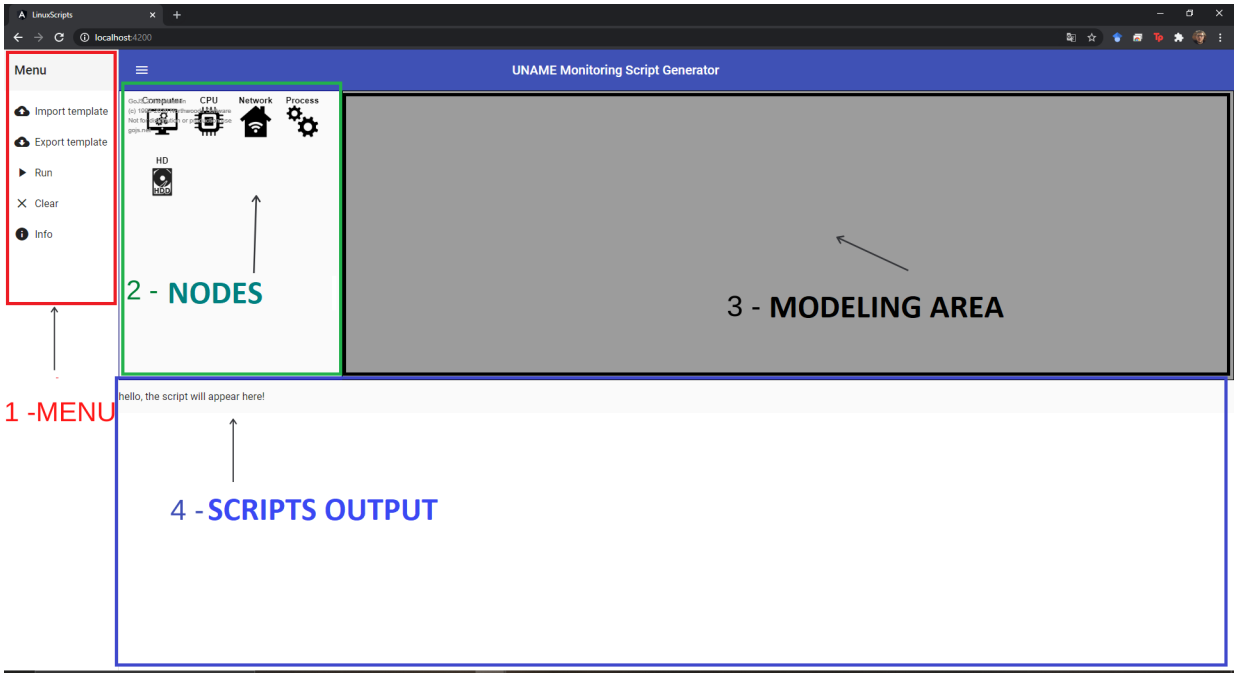

Figure 1. UNAME tool's main screen

TABLE I
NODES AND MONITORABLE RESOURCES

Node	Monitorable Resources
Computer	Date, Hour, Mem. Buffer, Mem. Cache, Mem. Free, Mem Shared, Mem. Total, Mem. Used, Swap free, Swap total, Swap used, Zombie Process
CPU	Core, Gnice, Guest, Idle, IOwait, Irq, Nice, Soft, Steal, Sys, User
Network	Download KB, Download Packet, Upload KB, Upload Packet.
HD	Blocks, Free Kb, Free Percente, Total, Used KB, Used Percent.
Process	Date, Hour, CPU, Physical Memory Percent, Resident Memory KB, Virtual Memory KB

type of monitoring, which can be local or remote. If it is local, you do not need any additional information, but if it is remote, it is necessary to specify the target machine's IP address, as well as the user and password to access that machine through SSH protocol - this information is mandatory if the selected monitoring type is remote.

As can be seen, you can also specify the duration of the experiment. The monitoring duration can be either infinite or finite. If finite, another text field is opened so that the duration of the experiment can be added. If infinite, the script will be placed in an infinite loop, and the user needs to make a manual interruption. Finally, a monitoring frequency field is also presented, and the evaluator must define a value that should not be too high to lose important information or too low to impact the system's performance.

Finally, the **Clear** feature is used to erase all previously modeled information. Thus, the user can start modeling again without having to delete item by item.

The tool's output is showed in a web text editor called CKEditor, which adopts the *"What You See Is What You Get"* (WYSIWYG). Once the is started and when the *Clear* button is pressed, the exit area of scripts will be indicated by the phrase

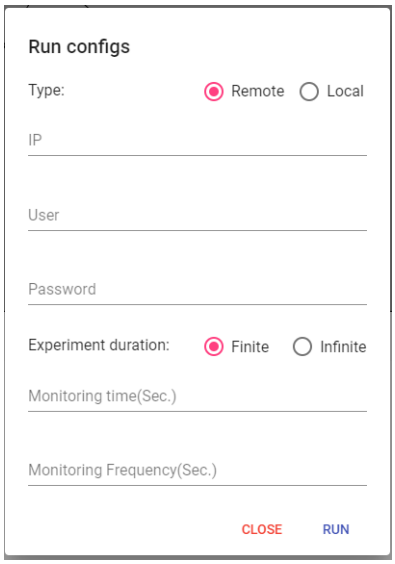

Figure 4. General Settings

"Hello, the script will appear here!". After configuring the nodes, we will have an output script as can be seen in Figure 5.

In the output script, it is possible to identify the properties that the user wants to monitor. When executed in a Linux terminal, a .txt file is created with the monitored information. In addition, a text editing area is also available, where it is possible to make style changes, customization or simply copy the script. The entire scope is delimited by creating the .txt file in the first *echo*, and ends at the *done* of the *sleep* command.

IV. Case Study

This section focuses on validating the proposed tool. For this, two different approaches were adopted: one based on the functionalities test, simulating user's iterations with the system through the Selenium Web Drive tool; and the second approach is based on experiment, monitoring real computational resources.

A. Validation #1: Selenium web

For this validation, 3 main steps are necessary. The first one is the assembly of pseudo-codes to compare with the output generated by UNAME tool; the second is to perform scenarios in the automated testing tool Selenium web drive; and the third step is the comparison of the tool's output with the pseudo-codes.

For each tested scenario, it is necessary to inform a title, detailed objective, preconditions, the steps to be followed and the expected result, which is the system's response to the actions performed.

Various system functionalities were tested, such as monitoring CPU resources, memory, disk, network, zombie processes, different sampling frequencies, and local and remote monitoring. However, due to space limitations, we will present a single test with a simple memory monitoring script.

The pseudo-code 2 presents the test scenario. **Title:** Validate used memory monitoring script generation
Detailed objective: Generate local monitoring script for computer nodes with infinite time. The resource to be monitored is used memory, with a sample frequency of 60 seconds.
Precondition: Structure assembled from at least one computer.
Steps:

1) Build Structure
2) Configure the node for the chosen property
3) Configure general script options

Expected result:

Algorithm 2: Pseudo-code of the expected result

```
1 #/bin/bash
2 echo "MemUsed" > "log.txt"
3 while true do
4 |   echo $(mon-MemUsed) >> "log.txt"
5 |   sleep 60 ;
6 end
```

After running Selenium, the output of the UNAME tool was the monitoring script #1. If we compare it with pseudo-code 2, it is possible to observe that the script generated by the UNAME tool fulfilled all functions provided in the pseudo-code.

Listing 1. Monitoring script

```
#!/bin/bash
echo MemUsed > log.txt
echo Monitoring...
while [ True ]
do
  mem = free | grep Mem
  time = date --rfc-3339=seconds
  MemUsed = echo $mem | awk '{print $3}'
  echo $MemUsed $time >> log.txt
  sleep 60
done
```

B. Validation #2: Real experiments

The main goal of these experiments is to test the scripts generated by the UNAME tool in a real environment. For this type of validation, two different experiments were carried out, one with local monitoring and other with remote monitoring. The first aims to identify the consumption of local OS resources, such as CPU and memory. And the other experiment aims to identify the impact in the network resources caused by the remote monitoring.

In the first experiment, it was adopted the same script used in the previous subsection, whose main functionality is to monitor the total memory used by the operating system. However, while this script was running, we decided to monitor it to identify the impact it could causing on the system. The total duration of the experiment was 1000 minutes. The results were quite satisfactory, as can be seen in Fig. 6 and Fig. 7.

Figure 6 shows the percentage of CPU and memory used by the monitoring script. The CPU utilization varied throughout the experiment between 0% and 0.2%, while the memory utilization was constant with only 0.02%. Figure 7 shows the utilization of virtual memory and resident memory. Both types of memories were constant throughout the experiment, only with approximately 6.6 MB and 3.2 MB, respectively.

The second experiment also has a duration of 1000 minutes, and analyzed the use of network resources caused by the monitoring script. This time, we use a script that collects the memory throughput from a remote computer. It was collected the network throughput of download and upload. Fig. 8 shows obtained results. The amount of data transferred remained constant in 3.2 KB/s (download) and 4.1 KB/s (upload), with slight variations during the experiment.

Therefore, according to the results of the monitoring experiments, it is possible to state that the interference caused by the monitoring script generated by the UNAME tool is insignificant.

Menu

Import template

Export template

Run

Clear

Info

```
#!/bin/bash
echo Data Hour MemBuffers SwapFree SwapTotal SwapUsed NumZumbis CpuGnice CpuGuest CpuIdle DiskBlocks DiskFreeKb
DiskFreePercent > PC1.txt

echo Monitoring...
while [ True ]
do


mem = free | grep Mem
tempo = date --rfc-3339=seconds
swap = free | grep Swap

data = echo $tempo | awk '{print $1}'
hora = echo $tempo | cut -d -f2 | gawk 'BEGIN{FS="-"}{print $1}'
membuffers = echo $mem | awk '{print $6}'
swapfree = echo $swap | awk '{print $4}'
swaptotal = echo $swap | awk '{print $2}'
swapused = echo $swap | awk '{print $3}'
numzumbis = ps aux | awk '{if ($8~"Z"){print $0}}' | wc -l

cpu = `mpstat 1 1 | grep ALL | head -1`
cpugnice = echo $cpu | awk '{print $11}'
cpuguest = echo $cpu | awk '{print $10}'
cpuidle = echo $cpu | awk '{print $12}'

disk = df | grep /dev/sda1
diskblocks = echo $disk | awk '{print $2}'
diskfreekb = echo $disk | awk '{print (100-$5)}'
diskfreepercent = echo $disk | awk '{print $4/$2 * 100.0}'

echo $data $hour $memBuffers $swapFree $swapTotal $swapUsed $numZumbis $CpuGnice $CpuGuest $CpuIdle $DiskBlocks $DiskFreeKb
$DiskFreePercent >> PC1.txt

sleep 1
done
```

Figure 5. Output script

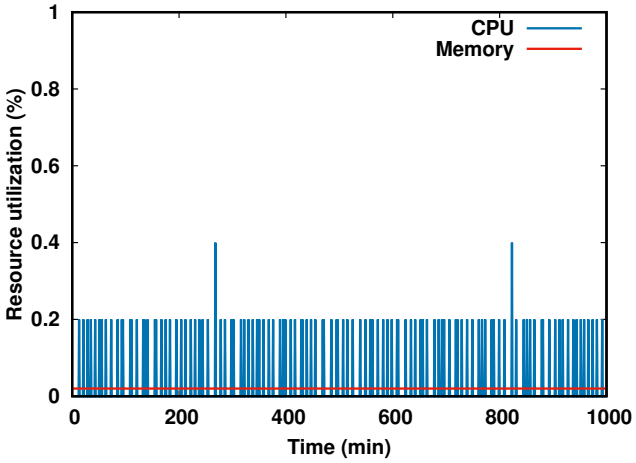

Figure 6. Percentage utilization

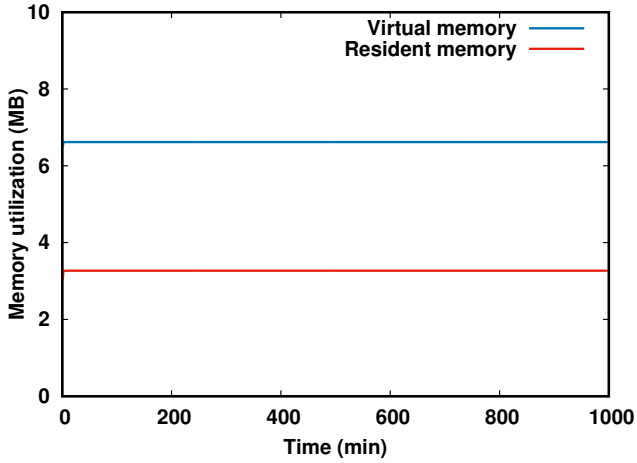

Figure 7. Memory usage

V. Related Tools

There is a non-exhaustive list of monitoring tools available, each one with its respective pros and cons. This Section presents a small set of these tools and compares them with the current version of the UNAME tool.

The Ganglia tool [13] is a scalable monitoring tool for distributed and high-performance computing systems (HPC), such as clusters and grids. It is a robust and complex tool, however, it fails to given the required support to evaluate the capacity and resources available by these kinds of systems.

Already the DRMonitor [14] is a distributed system for monitoring computational resources in a personal computer

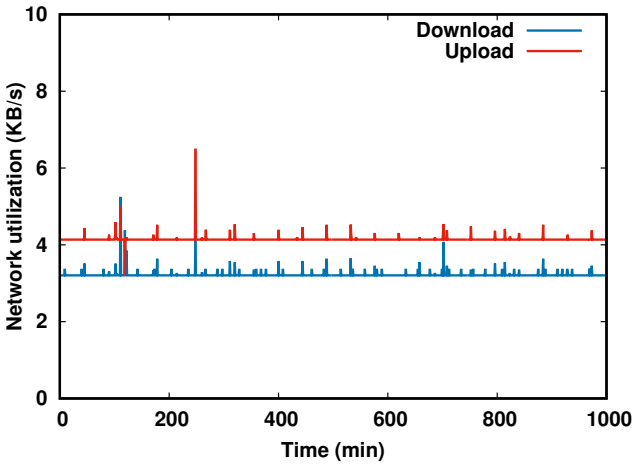

Figure 8. Network utilization

network. The system aims to assist applications that depend on workload information by periodically publishing updated metrics about the monitoring nodes. The DRMonitor allows monitoring performance metrics of a set of computers connected by a local network. However, this tool is already discontinued and does not have easy access to current repositories, making its use unfeasible.

There is also a homonymous tool that has a management interface developed in Java, but that uses Shell scripts to monitor resources. This new DR Monitor [15] allows the proactive scheduling and software rejuvenation actions for cloud computing system processes affected by the aging phenomenon. This system adopts monitoring scripts similar to those proposed by our tool but does not allow the personalized choice of resources to be monitored.

Finally, the MMONIT system [16] is a small open-source utility that, through command lines, allows you to manage and monitor Linux systems. It also performs automatic maintenance and repairs and can perform significant causal actions in error situations. This software monitors files, directories, file systems, and processes. Also, it provides a web interface for monitoring results. However, its main limitation relies on the need to purchase a license in order to use some of its locked features.

VI. Final Remarks

This paper presented the UNAME tool, a tool that provides automatic generation of Shell scripts that can be used to monitor either hardware or software resources. The proposed tool can be used from any browser but the generated scripts can be used to evaluate only Linux-based operating systems or that has Linux Bash installed, such as the Windows WSL environment.

We also presented a case study and an evaluation scenario that demonstrated the feasibility of the proposed framework. By pointed out how the UNAME tool can be used by both academics and industry to obtain the best information possible about their systems and setups with the minimum impact to systems performance, which is a result of a non-intrusive approach.

As future work, we intend to improve the UNAME tool to perform more than scripting generation, but also to perform real-time monitoring and present graphs and recommendations about the environment to system administrators, which can be done through an online dashboard.

References

[1] D. J. Lilja, *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

[2] M. Torquato, H. Mello, L. Torquato, J. Araujo, and E. Guedes, "Utilização de scripts para monitoramento de sistemas linux: Abordagem para criação de relatórios e gráficos com ferramentas open-source," in *Workshop de Software Livre Bahia-Alagoas-Sergipe (FreeBASE 2015), em conjunto com a XV Escola Regional de Computação Bahia-Alagoas-Sergipe (ERBASE 2015)*, 04 2015.

[3] J. Araujo, V. Alves, D. Oliveira, P. Dias, B. Silva, and P. Maciel, "An investigative approach to software aging in android applications," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 1229–1234.

[4] J. Araujo, R. Matos Junior, P. Maciel, , and R. Matias, "Software aging issues on the eucalyptus cloud computing infrastructure," in *Proceedings of the IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'11)*, Anchorage, USA, 2011.

[5] C. Melo, J. Araujo, V. Alves, and P. Maciel, "Investigation of software aging effects on the openstack cloud computing platform," *Journal of Software*, vol. 12, pp. 125–137, 02 2017.

[6] M. Torquato, J. Araujo, and P. Maciel, "Estudo experimental de envelhecimento de software em nuvens kvm/opennebula: Live migration como mecanismo de suporte ao rejuvenescimento de software," in *XIII Workshop de Computação em Clouds e Aplicações (WCGA 2015), em conjunto com o 33º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2015)*, 05 2015.

[7] R. Matos, J. Araujo, V. Alves, and P. Maciel, "Characterization of software aging effects in elastic storage mechanisms for private clouds," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*. IEEE, 2012, pp. 293–298.

[8] H. J. Syed, A. Gani, F. H. Nasaruddin, A. Naveed, A. I. A. Ahmed, and M. Khurram Khan, "Cloudprocmon: A non-intrusive cloud monitoring framework," *IEEE Access*, vol. 6, pp. 44 591–44 606, 2018.

[9] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York: Wiley Computer Publishing, John Wiley & Sons, Inc., May 1991.

[10] D. Taylor, *Wicked Cool Shell Scripts: 101 Scripts for Linux, Mac OS X, and Unix Systems*. No Starch Press, 2004.

[11] W. Stepniak and Z. Nowak, "Performance analysis of spa web systems," in *Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part I*, L. Borzemski, A. Grzech, J. Świątek, and Z. Wilimowska, Eds. Cham: Springer International Publishing, 2017, pp. 235–247.

[12] R. Blum, *Linux Command Line and Shell Scripting Bible*. Wiley Publishing, Inc, May 2008.

[13] M. Massie, B. Li, B. Nicholes, V. Vuksan, R. Alexander, J. Buchbinder, F. Costa, A. Dean, D. Josephsen, P. Phaal, and D. Pocock, *Monitoring with Ganglia*, 1st ed. O'Reilly Media, Inc., 2012.

[14] P. Domingues, L. Silva, and J. G. Silva, "Drmonitor - a distributed resource monitoring system," in *Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2003. Proceedings.*, Feb 2003, pp. 127–133.

[15] J. Araujo, C. Braga, J. Neto, A. Costa, R. Matos, and P. Maciel, "An integrated platform for distributed resources monitoring and software aging mitigation in private clouds," *Journal of Software*, vol. 11, pp. 976–993, 10 2016.

[16] MMonit. (2020) Easy, proactive monitoring of processes, programs, files, directories, filesystems and hosts — monit. [Online]. Available: https://mmonit.com/