Implementation of an email-based alert system for large-scale system resources

Robert Poenaru[1,2]
[1]*Horia Hulubei* National Institute of Nuclear Physics and Engineering, Magurele, Romania
[2]Doctoral School of Physics, University of Bucharest, Romania
robert.poenaru@nipne.ro

***Abstract*—Tackling the current problems of interest for physicists that deal with various topics require lots of computing simulations. Identifying and preventing any unusual behavior within the system resources that execute large-scale calculations is a crucial process when dealing with system administration, since it can improve the run-time performance of the resources themselves and also help the physicists by obtaining the required results faster. In the present work, a simple *pythonic* implementation which 1) monitors a given computing architecture (i.e., its system resources such as CPU and Memory usage), and 2) alerts a custom team of administrators via e-mail in almost real-time when certain thresholds are passed, is presented. Using existing packages written in Python, with the current implementation it is possible to send e-mails to a predefined list of clients containing detailed information about any machine running outside the "normal" parameters.**

***Index Terms*—python, system resources, alerting, email, smtp, monitoring, watchdog**

I. Introduction

The computing resources within a physics department must be up to speed and ready for a continuous run-time of small-, medium-, but also large-scale simulations, in order to assure a consistent and optimal workflow for the research teams that require calculations. Usually, there is a cohesive workflow between the scientists that want to run their simulations and the system administration (sysadmins) team that provides the necessary resources for executing them. The sysadmins must check that the resources which are performing calculations behave well, but they must also take care of the computing equipment that is sitting in *idle*-mode, in case new requests for allocating resources are issued by scientists. The process of resource management is crucial since it involves many factors in deciding which are the most optimal compute nodes that could start a new job, in terms of efficiency and speedup [1], these being deciding factors in the total run-time of the simulations themselves.

Proper job allocation, management, and execution will result in minimal impact of resource slowdown, process blocking, or even dead-locks in the executing pipeline, giving the possibility of the researchers to obtain the desired numerical results in as fast as possible.

On the other hand, any code optimization [2], [3] on the submitted simulations (done exclusively by the scientific teams) will also take advantage of the allocated resources, since *better code* implies faster execution, lower impact on the memory pool and much lower probability of program interruption.

One can conclude that indeed, better resource management (done by the sysadmins) will result in better code execution, helping thus the scientists, but in the same way, any code optimization made by scientists will help the sysadmins, since the degree of failure within the executing simulations stack could be decreased. This reciprocal-mode of improvements (for both *communities*) is sketched in Fig. 1, where the key characteristics of both *communities* (i.e., physicists issuing simulations and sysadmins dealing with computing management) are emphasized. The arrows signify the *reciprocal improvement cycle* between the two.

However, due to the large degree of complexity of the underlying computing infrastructure, issues related to memory bandwidth, network stability, CPU throttling [4], cache availability [5] and so on are highly probable, especially when the machines are running continuously. Frequent updates, unexpected network traffic, errors within the services running in the application layer [6] could also affect the idling nodes. While the former issues will only affect the simulations that are currently being executed, the latter set of issues could produce unexpected delays in the starting process of the job queue [7], which will increase the wait-time of simulation results. One can see that indeed, any issues which occur with the computing resources, even the idling ones, can affect the workflow of the research teams, causing the entire research department to finish their projects. As such, the sysadmins are essential by taking the proper actions on the computing infrastructure.

Unfortunately, there will always be moments when the sysadmin team that is monitoring a particular computing node (e.g., cluster, server racks, etc.) cannot keep track of the *health status* of the entire architecture at all times (although recently, some interesting models emerged within the literature that could improve the machine health status [8], [9] in an automated way). In order to properly, securely and efficiently maintain a large-scale server infrastructure up and running (as the one that is present in a physics research department), there must be some kind of *alert system* that is constantly analyzing the system resources, and tries to identify unusual behavior. When a potential malfunction is identified, the core-feature of the alert system is to inform the sysadmins with regards to the occurring issue(s).

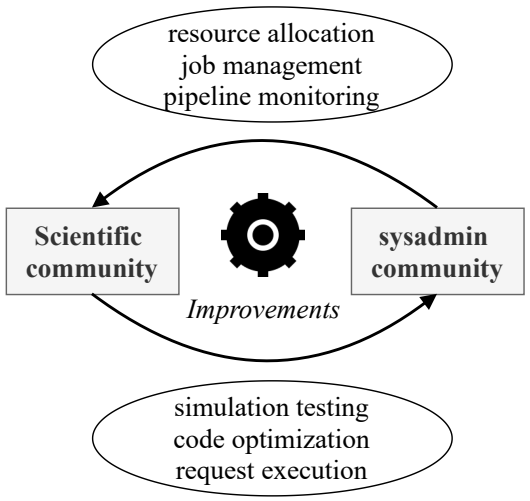

Fig. 1. A basic relationship between the scientific community that issues simulations to be executed on the computing resources, and the system administration team that deals with process allocation, execution and management of resources. The continuous improvements between the two communities involve the steps indicated next to the arrows.

In the present work, such a model is implemented; namely, a service that analyzes the logs generated by the system resources of a computing cluster (i.e., CPU usage, RAM usage and so on) and compares the values with predefined thresholds decides if the monitored values indicate some unusual behavior. If indeed, the thresholds are exceeded, the service will immediately raise an alert, sending information with the occurring issue to the sysadmin teams, which then can take action.

The paper is organized as follows: in Section I, a brief introduction of system administration is made, with motivation for implementing an alert system within a large-scale computing infrastructure. Furthermore, in Section II the general overview on how the *pythonic* implementation of the current research actually works, mentioning its main features and advantages. Section III discusses the services that are used in order to keep track of changes within the log-files generated by the system resources, while Section IV presents the method of deciding between *normal* and *unusual* behavior of said resources. Finally, an overview of the implementation which sends the actual alerts to the sysadmin team (via email) is made in Section V, followed up by some concluding remarks and potential improvements of the project itself, given in Section VI.

II. Alert-System Workflow

Implementation of the alert system is quite straightforward, following a procedure that does not require too much information. Fig. 2 shows how the process of taking action (i.e., fixing occurring issues on the computing nodes) by the administration team is taking place. Usually, the personnel that deals with system administration is located on-site, since the need-of-action would sometimes require physical access on the barebone servers. However, considering the current pandemic situation [10], the paradigm has shifted a lot to remote work, with possibility of remote access by the sysadmins [11] to any computing cluster located in the departments they monitor. As a result, the alert system will eventually inform both groups, and depending on the type of issue encountered, actions will be made by one or the other (or even both).

The reason behind choosing *Python* as the development tool here lies in its overall great compatibility with different operating systems, consistency between different system architectures (e.g., x86, ARM, [12]), well-documented resources, robust packages that allow the implementation of all the required instructions in a secure and reliable way.

Between the monitoring phase and the alert + action phases, the current approach involves three major stages:

Step0 - Service configuration
: This is done before the alert system is turned on, by setting some parameters to the desired values. The sysadmin team must decide on their values, depending on the required regime of execution. Discussion on the parameters will be made throughout the paper.

Stage-1 - *Data ingest*
: This is the first step, in which the implemented service is extracting the data coming from the computing resources. These data contain information with regards to the state of each resource, status of all the running processes and so one. They usually come in form of *log files*. Incoming data is stored at a fixed path, and the alert service will check for changes within those files.

Stage-2 - *Log analysis*
: With the incoming data from the monitored machines finally read, the service will analyze them in terms of their numerical values, i.e., for each system stat (CPU usage, RAM usage, incoming/outgoing network traffic). The stats of interest will be compared with the *thresholds*, which dictate whether the resources behave under normal or unusual conditions.

Stage-3 - *Alerts*
: If the monitored resources show that indeed their behavior is unusual (considering any of the controlled stats), the python implementation will trigger a so-called *alert*: a chain of procedures that follow the set of parameters defined in *Step0* which will basically inform sysadmins with regards to the encountered issue. The process of informing the sysadmin teams is done via email [13]. Its ease-of-use and ease-of-access allows the members to quickly and efficiently be informed, since the e-mail clients can run on pretty much any device (from a personal computer, to a laptop, tablet and smartphone), becoming a great solution in the current work.

Now that the core stages of the implementation were sketched (see also Fig. 3 for a graphical representation that encapsulates its features), it is instructive to describe each stage in particular, with its main goals and characteristics.

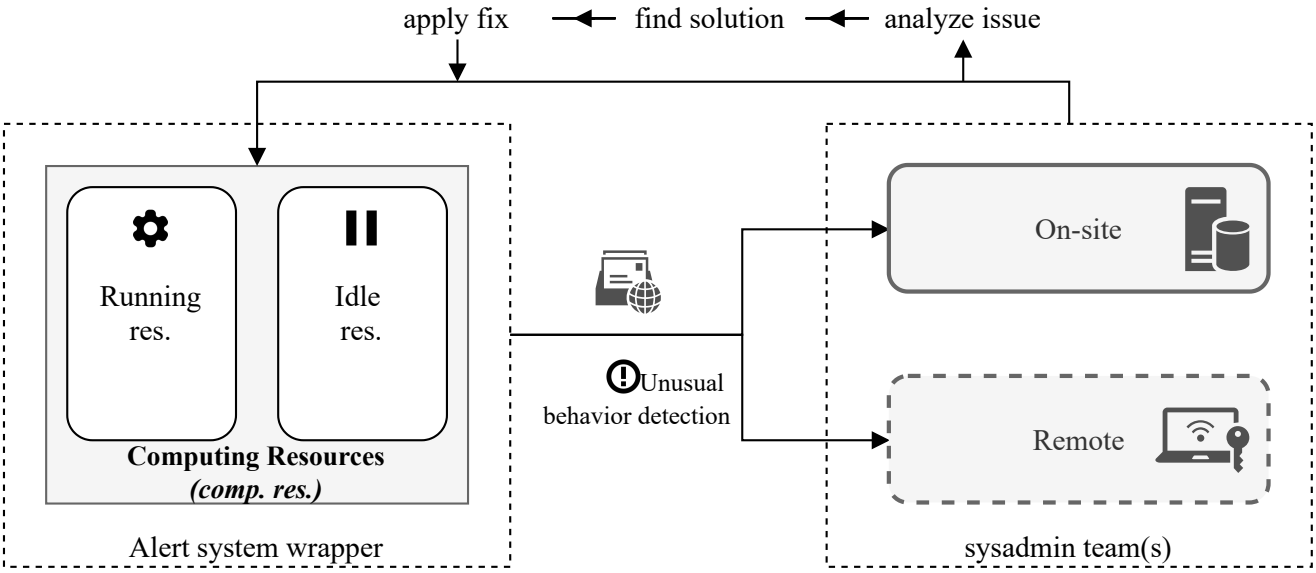

Fig. 2. The general workflow of a system administration team, when an alert system would be deployed on the computing resources that need to be monitored.

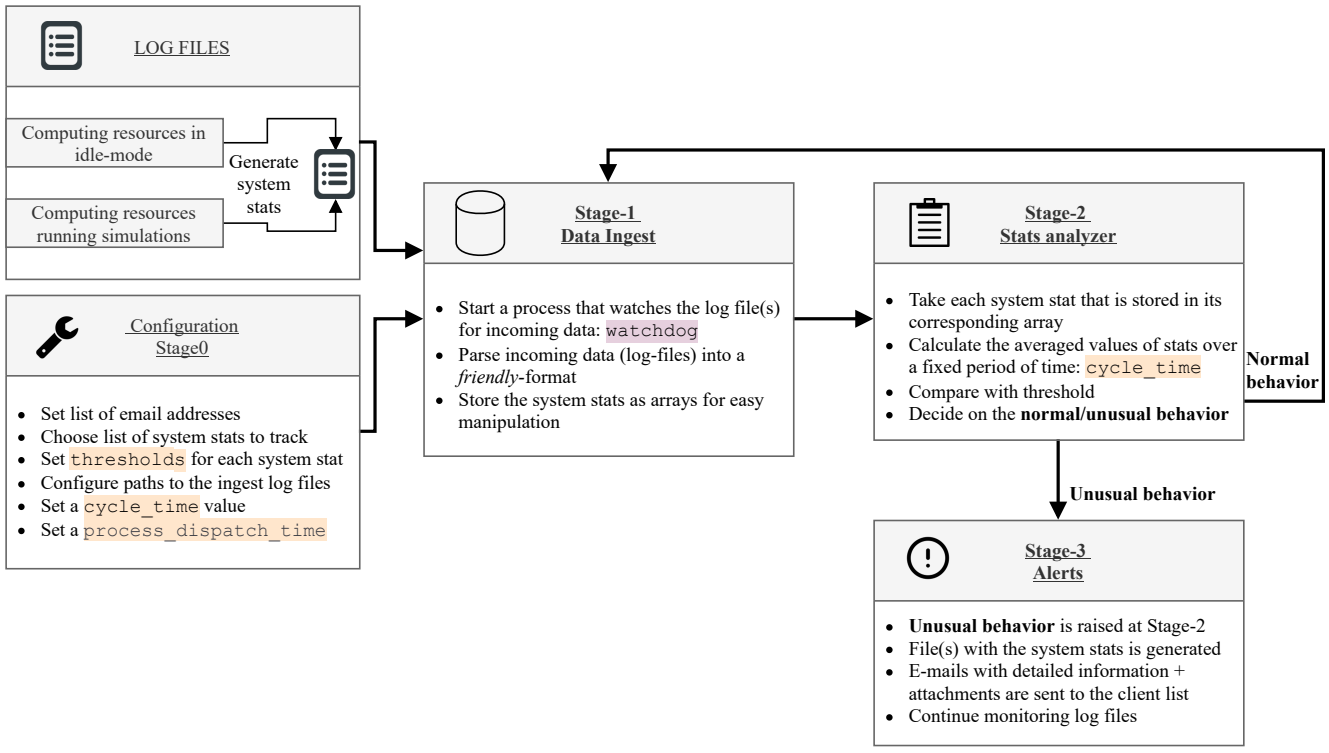

Fig. 3. Complete overview of the 3-stage operation mode of the alert implementation. Details on the marked variables are describe throughout the paper.

III. Watchdog - Data ingest

IV. Thresholds - normal vs. unusual behavior

V. Alert via email

VI. Conclusions & Future Work

Acknowledgment

This work done within the Department of Computational Physics and Information Technology at *Horia Hulubei* National Institute of Physics and Nuclear Engineering. The development was funded by European Regional Development Fund through project CECBID-EOSC (POC/397/1/1-124405).

References

[1] Ashkan Paya. Resource management in large-scale systems. *Department Of Computing Science Umea University*, 2015.

[2] PVS Studio. Code optimization, 2017. Last accessed 24 September 2021, `https://pvs-studio.com/en/blog/terms/0084/`.

[3] Krishna Sai. Code optimization, 2018. Last accessed 24 September 2021, `https://www.techtud.com/`.

[4] IBM Corporation. Cpu throttling, 2012. Last accessed 24 September 2021, `https://www.ibm.com/docs/en`.

[5] Caching challenges and strategies. Last accessed 24 September 2021, `https://aws.amazon.com/builders-library/`.

[6] Pstree - linux man page. Last accessed 24 September 2021, `https://linux.die.net/man/1/pstree`.

[7] IBM Corporation. Job queues, 2015. Last accessed 24 September 2021, `https://www.ibm.com/docs/en`.

[8] Ramin Hasani, Guodong Wang, and Radu Grosu. A machine learning suite for machine components' health-monitoring. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019.

[9] Chia-Yu Lin, Tzu-Ting Chen, Li-Chun Wang, and Hong-Han Shuai. Health-based fault generative adversarial network for fault diagnosis in machine tools. *The 4th International Workshop on Artificial Intelligence of Things*, 2020.

[10] Marco Ciotti, Massimo Ciccozzi, Alessandro Terrinoni, Wen-Can Jiang, Cheng-Bin Wang, and Sergio Bernardini. The covid-19 pandemic. *Critical reviews in clinical laboratory sciences*, 57(6):365–388, 2020.

[11] Amira B Sallow, Hivi Ismat Dino, Zainab Salih Ageed, Mayyadah R Mahmood, and Maiwan B Abdulrazaq. Client/server remote control administration system: Design and implementation. *Int. J. Multidiscip. Res. Publ*, 3(2):7, 2020.

[12] Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12. IEEE, 2013.

[13] Jonathan Postel. Rfc0821: Simple mail transfer protocol, 1982.