Turning Privacy Constraints into Syslog Analysis Advantage

Siavash Ghiasvand
Technische Universität Dresden, ZIH
01069 Dresden, Germany
siavash.ghiasvand@tu-dresden.de

Florina M. Ciorba
University of Basel, Department of Mathematics and Computer Science
4001 Basel, Switzerland
florina.ciorba@unibas.ch

Wolfgang E. Nagel
Technische Universität Dresden, ZIH
01069 Dresden, Germany
wolfgang.nagel@tu-dresden.de

Nowadays, failures in high performance computers (HPC) became the norm rather than the exception [10]. In the near future, the mean time between failures (MTBF) of HPC systems is expected to be too short, and that current failure recovery mechanisms e.g., checkpoint-restart, will no longer be able to recover the systems from failures [1]. Early failure detection is a new class of failure recovery methods that can be beneficial for HPC systems with short MTBF. Detecting failures in their early stage can reduce their negative effects by preventing their propagation to other parts of the system [3].

Linux, with a share of 97%, is the dominant operating system among world top 500 supercomputers [11]. All Linux-based systems, support Syslog [4] as a basic service. By default vital parts of the system send their activity log into Syslog service. Therefore, Syslog entries (hereafter, *syslogs*) are invaluable sources of information to monitor and analyze system behavior. Because of Linux popularity, any approach based on syslogs can be easily extended to be used on other HPC systems. Also, syslog analysis is a passive approach and, therefore, it has no influence on the system performance. In [6] we proposed an approach to predict failures. Later, in [5] we revealed the temporal and spatial correlation of failures. And in [7] we introduced a prototype to analyze system behavior. This body of research became possible solely because of the existence of the HPC system (*Taurus*[1]) syslogs.

Syslogs contain very detailed information about all fundamental services and daemons. On one hand, this detailed information helps understanding the system behavior better. On the other hand, too much data makes it harder to extract the required information. Apart from these pros and cons, syslogs also contain sensitive data which might be used to uniquely identify system users and to track their activities.

[1] https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus

Even though there were some projects in the past [9, 8] to publish syslogs, concern about users privacy prevents system administrators to actively participate in such projects, and publish their HPC system syslogs. Which consequentially limits us to our local sources of syslogs, and prevents us from proofing and generalizing our failure prediction approaches. The goal of the current work is to contribute to the foundation of failure detection techniques via sharing an ongoing research with the community. Herein we consider user privacy as the main priority, and then turning the applied constraint for protecting users privacy into an advantage for analyzing the system behavior. We use De-identification, constantification, and hashing to reach this goal. Our approach also contributes to the reproducibility and openness of future research in the field. Via this approach, system administrators can unwarily share their syslogs with the public domain.

This is how a typical syslog entry looks like:

```
1 Session opened for user siavash by (uid=0)
2 Received disconnect from 192.168.32.21: 10:
3 Session opened for user florina by (uid=0)
4 Session closed for user siavash
5 Accepted key for siavash from 192.168.12.31
6 Syslog-ng starting up; version='2.1.1'
```

In lines 1, 3, 4, and 5 *username* and in lines 2 and 5 *IP address* of users are visible. Both are unique identifiers and by tracking these identifiers along the syslogs we can monitor the specified user's activity. In line 1 and 3, except the *usename* and *uid*, the rest of entries are identical. Previous findings [5, 7, 6] indicate that even though the detailed information is useful to traceback the root of system misbehaviors, for keeping track of syslog patterns, and failure prediction, they do not yet have an added value. Therefore, by removing sensitive information from the above mentioned syslogs, we can safely eliminate the concerns about users privacy.

```
1 Session opened for user USER by (uid=0)
2 Received disconnect from IP: 10:
3 Session opened for user USER by (uid=0)
4 Session closed for user USER
5 Accepted key for USER from IP
6 Syslog-ng starting up; version='2.1.1'
```

So far, we de-identified the syslogs (hereafter, *D-syslogs*). Now in favor of syslogs readability we can continue to replace variable parts with constant strings. Too much data

in this level for our purpose, only results in consuming more resources and does not provide any added value.

```
1 Session opened for user USER by (uid=UID)
2 Received disconnect from IP: SESSION:
3 Session opened for user USER by (uid=UID)
4 Session closed for user USER
5 Accepted key for USER from IP
6 Syslog-ng starting up; version='VERSION'
```

After constantifying our D-syslog sample, line 1 and line 3 became identical. As a real test case, via full constantification of *Taurus* D-syslogs, we observed that out of 750 million entries there are only less than 1000 unique entries.

We already saved a lot of space and processing time, via de-identifying and constantifying syslogs (hereafter, *DC-syslogs*). Hashing, brings us to the next level. Using the hashed entries, rather that the raw text, significantly increases the efficiency of pattern matching algorithms [2]. Hashing the DC-syslog sample, gives us the below output:

```
1 4D78CB0D020FDE1D3062F953511C8ACD
2 0A94C7C3CC8B5551BC637031EAE4E91F
3 4D78CB0D020FDE1D3062F953511C8ACD
4 26B87F76CEC196EE0D355EEC47AF8216
5 A95DAD54FC6A7E5DE1EF5D63D6C58DA0
6 A15ADB95404F0CF3875DCACA828A63FB
```

The complete set of hash codes on our HPC machine has less than 1000 elements. Therefore using a 3 digit hash code is sufficient, and we can save even more space. The final pre-processed syslog sample looks like this:

```
1 001
2 002
3 001
4 003
5 004
6 005
```

Note that the first and third line were given the same 3-digit hash code. For some pattern matching algorithms a constant hash code length is more beneficial.
We ran the very same approach on *Taurus*. On average we enjoyed from more than 90% disk space reduction. Using the proposed approach, even though we have to perform a pre-processing of syslogs, the overall performance is at least 3 times higher than processing the plain unaltered syslogs.

The big picture and conclusion
Analyzing syslogs, gives us a deep insight about system behavior. Our preliminary results indicate that there are patterns in syslogs. Using these patterns we can detect failures in their early stage, and ideally predict them. Beside normal information, syslogs contain sensitive data which can potentially enable syslog analyzers to endanger users privacy. Because of these privacy concerns, there are very few publicly accessible collection of syslogs. On the other hand, too many details in syslogs, makes their storage expensive, increases the analysis complexity, and hides the useful information. De-identifying, constantifying, and hashing syslogs (hereafter, *DCH-syslogs*), address both challenges. The DCH-syslogs, have no sensitive data, need less disk space, take less time to analyze, and give a clear view over the system behavior without hiding the important information. Since DCH-syslogs, do not contain any sensitive data and take less disk space, they can freely and easily be released to the public domain. By using DCH-syslogs, we are intentionally ignoring some details in the data. These details do not have interesting information for our failure prediction approach. The benefits gained, e.g. diverse syslogs from different HPC machines to test and prove the efficiency of prediction mechanisms, faster analysis, reproducible researches, contributing to open science, and so on, outweight the costs.

1. REFERENCES

[1] F. Cappello, A. Geist, and W. Gropp. Toward Exascale Resilience: 2014 update. *Supercomputing Frontiers and Innovations*, 1(1):5–28, 2014.

[2] R. M. Cowan and M. L. Griss. Hashing, the key to rapid pattern matching. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, pages 266–278, London, UK, UK, 1979. Springer-Verlag.

[3] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. Failure prediction for HPC systems and applications: Current situation and open issues. *International Journal of High Performance Computing Applications*, 27(3):273–282, July 2013.

[4] R. Gerhards. RFC 5424, 2009.

[5] S. Ghiasvand, F. M. Ciorba, and W. E. Nagel. Toward Resilience in HPC: A Prototype to Analyze & Predict System Behavior. *International Supercomputing, poster*, 2016.

[6] S. Ghiasvand, F. M. Ciorba, R. Tschüter, and W. E. Nagel. Analysis of Node Failures in High Performance Computers Based on System Logs. *Supercomputing, poster*, 2015.

[7] S. Ghiasvand, F. M. Ciorba, R. Tschuter, and W. E. Nagel. Lessons Learned from Spatial and Temporal Correlation of Node Failures in High Performance Computers. *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 377–381, 2016.

[8] Google. www.github.com/google/cluster-data, 2011.

[9] LANL. www.usenix.org/cfdr-data, 2005.

[10] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. Debardeleben, P. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. S. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen. Addressing failures in exascale computing. *International Journal of High Performance Computing*, 2013.

[11] Top500. www.top500.org, 2016.

Source code
We are using Python[2] scripts to analyze syslogs. The Python scripts used in this work can be freely downloaded[3]. Please note that these script are only a proof of concept, and released solely to support better understanding of this work. Therefore, they are kept as simple as possible.

[2] `https://www.python.org/`
[3] `http://wwwpub.zih.tu-dresden.de/~ghiasvan/publications/sc16/`