In 7th Conference on the Engineering of Computer Based Systems (ECBS 2021), May 26–27, 2021, Novi Sad, Serbia. ACM, New York

Towards Automatic Application Fingerprinting Using Performance Monitoring Counters

Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson
first.last@mdh.se
Mälardalen University
Västerås, Sweden

Marcus Jägemar
marcus.jagemar@ericsson.com
Ericsson AB
Stockholm, Sweden

ABSTRACT

In this paper, we discuss a method for application fingerprinting using conventional hardware and software performance counters. Modern applications are complex and often utilizes a broad spectra of the available hardware resources, where multiple performance counters can be of significant interest. The number of performance counters that can be captured simultaneously is, however, small due to hardware limitations in most modern computers. We propose to mitigate the hardware limitations using an intelligent mechanism that pinpoints the most relevant performance counters for an application's performance. In our proposal, we utilize the Pearson correlation coefficient to rank the most relevant PMU events and filter out events of less relevance to an application's execution. Our ultimate goal is to establish a comparable application fingerprint model using performance counters, that we can use to classify applications. The classification procedure can then be used to determine the type of application's fingerprint, such as malicious software.

ACM Reference Format:
Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson and Marcus Jägemar. 2021. Towards Automatic Application Fingerprinting Using Performance Monitoring Counters. In *7th Conference on the Engineering of Computer Based Systems (ECBS 2021), May 26–27, 2021, Novi Sad, Serbia.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3459960.3461557

1 INTRODUCTION

An application execution is complex and requires many hardware resources. The way program utilizes hardware resources reflects a distinct fingerprint that describes the execution-time behavior of an application. Many system architectures are deployed as off-the-shelf solutions so such fingerprint can provide an additional layer of understanding for engineers, enabling them to fine-tune the resource allocation, scheduling processes, identifying threats and protect their IT infrastructures.

Modern computers often implement a performance monitoring unit (PMU) that can measure the system hardware resource usage. The PMU implements a large set of events, measuring for example resources in the CPU pipeline, various internal memory events such as cache events, and also off-core events including system-level caches, and main memory accesses. The PMU utilizes hardware-implemented performance monitoring counters (PMC) that count the number of occurrences of a certain event during a time interval. Modern computers typically implement a few PMCs to enable simultaneous measurement of multiple events.

Several interfaces in Linux provide access to the PMU, such as PAPI [4], and perf [2]. Such tools are triggered by manual configuration at the command-line and for a very limited set of events at a time. Some tools try to expand the number of simultaneously measured events through time-based multiplexing but this could cause a significant overhead limitation when the measurement frequency is small. There is also a possibility to measure performance counters at only certain time points of an application's execution. For instance, we can measure one set of hardware events for the first half of an application's execution and then swap with another set of hardware events for the other half. Splitting the hardware events measurements over the timespan of an applications, however, may not reflect the true start-to-end behavior and resource dependence of an application.

Moreover, a big challenge is to envelop all platforms because a substantial number of hardware events are native to a specific platform. It is possible that fingerprinting might provide a successful on one hardware that contains the specific performance counters events, while on other hardware it is not possible since it does not contain the required events.

Other researchers have employed performance counters for various purposes such as monitoring hardware capacity, application performance, system health-check, and for detection purposes. Many of these studies are limited to a pre-selected set of hardware events. Jägemar et al. [3] proposed a service associated with CPU scheduler for an improved QoS through performance monitoring counter measurements. Danielsson et al. [1] used performance monitoring counters to identify resource dependence of an application in multi-core system and to see if the application needs last-level cache partition slices. In our opinion, a fingerprinting service should be able to automatically the most relevant hardware events for the application under observation because it is not always possible to have such knowledge beforehand. Inspired by these studies [1, 3], our goal here is an automatic framework to:

- Monitor process hardware resource usage by utilizing the Performance Monitor Unit (PMU) for a large set of events.
- Find the hardware resources which have the highest impact on process performance among a large set of PMU events.
- Automatically create a hardware usage fingerprint (model) that represents the process' hardware usage.

2 CHALLENGES

Our prime challenge is to combat the shortage of physical counters in contrast to monitoring the hundreds of performance counter events. The general capacity of modern computers can vary, a typical Intel Core processor can for instance host around 3 fixed-function and 4 general-purpose counters per PMU. However to justly profile an application, significantly more events than four are required.

Secondly, to target numerous platforms with an identical set of events is irrational due to the fact that hardware events depend on the underlying architecture and hence differ from platform to platform. However, there are many events which are common across various platforms. An approach able to confront the differences in performance counter events and between different hardware would render a flexible and cross-platform service to users.

Thirdly, profiling the programs with short execution time is itself challenging. Approaches like event multiplexing, can aid in capturing more events for a time slice but may not be a viable solution for the programs with very short execution time. Even so, short execution time have advantages such as the possibility to frequently re-running the application and capturing different sets of events iteratively with less profiling time and performance degradation.

Moving forward towards characterization of requirements of the programs with busy-wait loop is another challenge. In principle, this is a typical case for embedded systems where functionality is periodically polled within a busy-wait loop. Therefore, to reveal the start-to-end hardware requirements of a never ending program is somewhat that can only be addressed by finding an optimal breaking point of measurements.

3 DISCUSSION

We suggest a mechanism that automatically samples n PMU events corresponding to some hardware resources monitored for the process p. Each sampled PMU event originates a time-ordered series, m_i. All series are collected in the set $M_{(p)} = \{m_i : 0 \leq i \leq n\}$ and, for each one of them, we calculate Pearson's correlation coefficient, r_i, between m_i and the measured performance of p. Finally, we define the fingerprint of p, $F_{(p)} \subseteq M_{(p)}$, by selecting a subset of the m_i series among those with highest r_i values. In this way, $F_{(p)}$ denotes the set of events that are more correlated to the performance of p, where $|F_{(p)}|$ depends on fingerprint storage capacity and desired quality.

The rational behind automatic sampling of n events is to ensure that the fingerprinting process is optimal for anonymous or previously unknown applications. We sample all *native events* provided by the underlying platform at 5ms granularity. Since the total execution time of application may vary between different runs, we sample the events with reference to *Retired-Instructions* which does not get affected by the computing environment for different instances. The captured statistics are then pruned with Pearson Correlation to determine strength and direction of each event in terms of start-to-end behavior of an application. Since the most relevant events in $M_{(p)}$ are to be determined automatically, the acquired measurements and ranking can be used for a sustainable model which is flexible enough to serve multiple purposes such as to maintain sufficient *QoS*, and for *Matching* profiles, as shown in Fig.1. There could be many *Matching* profiles but the idea is to enhance the mechanism into a *Detection* service which can uniquely identify the processes.

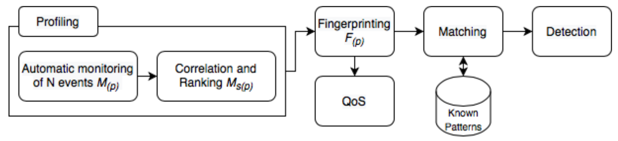
Profiling
Automatic monitoring of N events $M_{(p)}$
Correlation and Ranking $M_{s(p)}$
Fingerprinting $F_{(p)}$
QoS
Matching
Known Patterns
Detection

Figure 1: Proposed solution for application fingerprinting

We measure PMU events of applications and initially without complete understanding on how the events correlates to an application's functional behavior. However, through careful analysis of the PMU along with the application's metadata, we can also build and refine a security profile to recognize potential security threats by comparing the current execution profile to previously created execution fingerprints.

In short, by automatically monitoring $M_{(p)}$ possible events, we propose an application *fingerprint* as a model that describes the execution profile of a process. Finally, the main purpose of the fingerprint in the context of this paper is to ensure that the hardware resource usage is correctly modeled and that the quality is sufficiently good to ensure that it is possible to detect such an application running at a later stage.

4 FUTURE WORK

A flexible and intelligent mechanism to make the fingerprinting process automatic and reliable requires an in-depth study. One of the future work directions can be to optimize the jittery measurements for more accuracy. Accuracy could also be influenced by malfunctioning of any hardware resource.

We are also investigating to come up with a single strategy which can work for programs with short execution time as well as long execution time. Because currently on one hand multiplexing is not possible for short programs and re-running is costly for long execution time programs.

An ideal solution would be to monitor more and more events regardless of platform dependence so that the detection service will be flexible enough to discern the emerging pattern. This feature can also make the fingerprinting service capable of revealing the disguised motives besides the expected behaviour.

REFERENCES

[1] Jakob Danielsson, Tiberiu Seceleanu, Marcus Jägemar, Moris Behnam, and Mikael Sjödin. 2020. Resource Depedency Analysis in Multi-Core Systems. In *44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 87–94.
[2] Brendan Gregg. 2014. *Systems Performance : Enterprise and the Cloud.* Upper Saddle River, NJ : Prentice Hall.
[3] Marcus Jägemar, Andreas Ermedahl, Sigrid Eldh, and Moris Behnam. 2017. A Scheduling Architecture for Enforcing Quality of Service in Multi-Process Systems. *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (2017), 1–8.
[4] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. 2010. Collecting Performance Data with PAPI-C. In *Tools for High Performance Computing 2009.* Springer, Berlin, Heidelberg, 157–173.