Mälardalen University Press Licentiate Theses
No. 347

AUTOMATED PERFORMANCE PROFILING OF SOFTWARE APPLICATIONS

Shamoona Imtiaz

2023

School of Innovation, Design and Engineering

ISBN 978-91-7485-610-1
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Abstract

For industrial systems performance, it is desired to keep the IT infrastructure competitive through the efficient use of computer resources. However, modern software applications are complex and often utilize a broad spectrum of available hardware resources. The way how applications utilize these resources may vary from platform to platform due to the different architectural features, requirements and performance levels guaranteed by the hardware as well as due to the type of application under analysis. It becomes challenging to predict how the deployed applications will perform on a particular platform, how to improve the hardware resource utilization, and how to meet the Quality of Service (QoS) requirements.

Computers these days enable us to precisely trace down the performance of applications using the Performance Monitoring Counters (PMCs) available in the Performance Monitoring Unit (PMU) of the processors. PMCs can record micro-architectural events, called PMU events, at the CPU cycle level. Tools like perf API and PAPI provide performance information using manual and selective function calls. Nevertheless, it is difficult for humans to make analyses, visualize performance over time and draw conclusions from this wealth of data without automatic and intelligent tools.

In this thesis, our first contribution is to propose a cross-platform automated approach to investigate the overall performance profile of the applications. Instead of relying on a static and pre-selected list of hardware and software performance events we avoid the selection bias by capturing the entire range of performance events specific to the platform on which applications are running.

The performance data being generated from shared resource environments and hierarchical resource utilization demands makes it harder to represent the

behavior in one model. That being the case, it was deemed appropriate to demonstrate the compact representation of behavior. So, our next contribution is to present a simplified model to understand the behavior of performance events. Therefore, we determine segments in performance data by locating the points in their data distribution using the change point detection method. The proposed solution reduces the complexity of data handling, allows the application of further statistical analyses and provides better visualization.

Lastly, to reveal the out-of-sight information, we present a customized approach to automatically identify the groups of similar performance events based on the change in their behavior. There can be several ways to group the performance data, we opt to form the groups based on change points in the behavior of the performance events. The knowledge can then be used by the decision-makers as per their interests such as for load balancing, deployments, scheduling and anomalous behavior detection.

Sammanfattning

Inom system som kräver hög prestanda är önskan att hålla IT-infrastrukturen konkurrenskraftig genom effektiv användning av datorresurser. Moderna mjukvaruapplikationer är komplexa och använder ofta ett brett spektrum av tillgängliga hårdvaruresurser. En applikations resursutnyttjandebeteende kanske inte är detsamma eftersom de olika hårdvara arkitekturerna, hårdvarustödet och mängden information en maskin kan bearbeta samtidigt kan variera från en plattform till annan. Det blir en utmaning att prediktera hur de utplacerade applikationerna kommer att prestera på en viss plattform, vad är det bättre utnyttjandet av hårdvaruresurser och hur man uppfyller kvalitetskraven (QoS).

Moderna datorer nuförtiden möjliggör att vi kan spåra applikationernas prestanda exakt med hjälp av Performance Monitoring Counters (PMC) som finns tillgängliga i processorernas Performance Monitoring Unit (PMU). Verktygen som perf API och PAPI tillhandahåller prestandainformation med hjälp av manuella och selektiva funktionsanrop. Ändå är det svårt för det människan att analysera, visualisera prestanda över tid och dra slutsatser från denna mängd data utan tillgång till automatiska och intelligenta verktyg.

I den här avhandlingen är vårat första bidrag att föreslå ett lösning som undersöker automatiskt applikationernas övergripande prestandaprofil. Istället för att förlita oss på en statisk och förvald lista över hårdvaru- och mjukvaruprestandahändelser undviker vi urvalsbias genom att fånga upp hela utbudet av PMU-händelser som är specifika för plattformen där applikationer körs.

Prestandadata som genereras från delade resursmiljöer gör det svårare att representera beteendet i en modell. Så vårt nästa bidrag är att presentera en förenklad modell för att förstå beteendet hos prestandahändelser. Därför föreslår

vi ett automatiserat tillvägagångssätt genom att segmentera prestandadata i mindre dataserier och tillhandahålla en statistisk modell för varje segment i stället för hela spektrumet. Den föreslagna lösningen ger fördelar som minskad komplexitet för datahantering, tillämpning av ytterligare statistiska analyser och bättre visualisering.

Slutligen, för att avslöja den osynliga informationen, presenterar vi ett anpassat tillvägagångssätt för att automatiskt identifiera grupper av liknande prestationshändelser baserat på förändringen i deras beteende. Det kan finnas flera sätt att gruppera prestationsdata, vi väljer att bilda grupperna baserat på förändringspunkter i beteendehändelserna. Kunskapen kan sedan användas av beslutsfattarna enligt deras intressen till exempel lastbalansering, schemaläggning och upptäckt av avvikande beteende.

To my family

Acknowledgment

Confucius formulated that *'We have two lives, and the second begins when we realize we only have one'*. Moving to Sweden and starting my Ph.D. was like leaving the uncertainty behind by empowering myself with higher education. So first of all, I would like to express my gratitude to my first teacher, my father, who taught me how to face life. Thank you **Abu G**, your lessons never get old.

Thanks to MDU and all my supervisors for providing me with the best-suited environment to achieve this goal. Thank you **Moris Behnam** for your continuous support and for fostering my passion for cybersecurity with various opportunities. You have the eye and openness to shape a talent. Thank you **Jan Carlson** for your instant and valuable feedback, you have been like a catalyst throughout my mentorship. Thank you **Gabriele Capannini** for inspiring this journey with your mathematical skills and problem-solving ideas, you always have a way forward. Thank you **Marcus Jägemar** for always appreciating small steps to achieve bigger goals, your encouragement brought me the confidence to seek bigger challenges.

I deeply appreciate **Jakob Danielsson** for your help in onboarding my Ph.D. journey. I would also like to express my sincere gratitude to **Robbert Jongeling** for being so kind, helpful and approachable in providing quick information than the internal portal. I am also thankful to my colleagues and peers for the cheerful times at Java for lunches and Fika.

Finally, I would like to extend my gratitude to my husband for holding me through the ups and downs of life, to my brother for bringing all the positivity in me, to all my siblings for always being supportive and to my parents for their unconditional love and hard work they had done to make us strong and good

individuals. Most importantly, to my kids, **Saleh** and **Abdullah**, to restore my energy on my way back home, You Are Life.

Shamoona Imtiaz
Västerås, October, 2023

List of Publications

Papers included in this thesis[1]

Paper A: Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jagemar. *Automatic Platform-Independent Monitoring and Ranking of Hardware Resource Utilization.* In the 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2021).

Paper B: Shamoona Imtiaz, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jägemar. *Automatic Segmentation of Resource Utilization Data.* In 1st IEEE Industrial Electronics Society Annual On-Line Conference (ONCON 2022).

Paper C: Shamoona Imtiaz, Gabriele Capannini, Jan Carlson, Moris Behnam, Marcus Jagemar. *Automatic Clustering of Performance Events.* In 28th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2023).

[1]The included papers have been reformatted to comply with the thesis layout.

Related publications, not included in this thesis

Paper X: Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jägemar. *Towards Automatic Application Fingerprinting Using Performance Monitoring Counters.* In the proceedings of 7th international Conference on the Engineering of Computer Based Systems (ECBS 2021).

Contents

I

Thesis

Chapter 1

Introduction

To reduce human intervention and efforts, technology-driven solutions are on the rise. This adherence to automation has led to rapid scaling of applications consequently transforming the world into a smaller place (a global village [1]) which is a complex digital system. This increased complexity of software and hardware architecture affects the efficiency of deployment, creates difficulties in predicting the system's behavior, makes it hard to optimize the systems against better usage of hardware, increases the power consumption, impacts the quality of service, influences the user experience and increases the cybersecurity risks. As a countermeasure to complexity, the community has proposed different approaches such as modularity [2, 3], abstraction, reusability [3], multilevel approach [4], standardization, and simplification [5]. These approaches are good design principles but achieving the desired functionality and better insight into hardware (multilevel cache, pipeline, internal busses, etc.) and software utilization is still a challenge due to complex interactions, diverse technologies, non-linear relationships, dynamicity, scale and size. To gather these insights from the designer and administrative perspective, load analysis and resource analysis are performed respectively by monitoring the performance of the systems [6].

System performance monitoring is categorized into processor utilization, memory usage, disk activity and network usage [6]. That implies it is related to the resource utilization demand of the software running over hardware. In

general, modern applications have complex resource utilization behavior such as in multicore systems a simultaneous demand for a shared cache can directly impact the system performance and can provide a variable user experience in a shared resource environment. To achieve the optimization goals it becomes interesting to know which hardware resources an application is utilizing the most, what hardware specifications can serve the demands efficiently, and how a resource demand may generate the request for the other resources. For such reasons, assessment of the expected performance is part of the performance monitoring routines. Detailed knowledge of hosted applications on a platform can forecast potential problems in addition to improved productivity. However, this information is not readily available without intelligent tools.

One way to precisely trace down the performance of applications is using Performance Monitoring Counters (PMCs) [7, 6, 8]). PMCs are hard-wired registers in Performance Monitoring Units (PMUs) of modern processors and are both architecture and vendor specific. These PMCs are responsible for monitoring micro-architectural PMU events at the CPU cycle level. The type and number of PMU events is dependent on the underlying architecture and so are the number of PMCs. Regardless of the architectural variations, the number of available PMCs is always significantly lower than the number of PMU events. These limitations restrict the engineers from monitoring more PMU events than the number of available PMCs. To them, it is a challenge to monitor numerous performance events without multiplexing many events on each PMC. Moreover, typical practices are static solutions based on a pre-selected list of PMU events with respect to engineers' interests and knowledge. This brings up the challenge of an increased likelihood of missing out-of-sight information. Nevertheless, such inconsistencies in the assessments can be reduced with the help of an automated cross-platform mechanism capable of determining the performance profile of the applications based on data-driven solutions instead of relying on general practices only. A cross-platform approach is what is not limited to a particular hardware architecture, is not restricted to some vendor specifications and does not require significant modification in the method when is applied across different platforms. This ultimately allows to make better decisions if engineers have the provision of such information.

In particular, our main goal was to analyze how the applications utilize

computer resources. Here the first deliberation was to avoid selection bias (also called Survivorship bias, an error identified by Abraham Wald)' [9] by capturing the overall behavior native to the platform on which the application is running. One of the challenges is that an application may not show the same behavior if it has been tested on a different platform and is running on a different one. Either way, measuring all returns a huge number of performance events. So our obvious goal was to 'look beyond the data in front of us' and funnel down the data from 'all' to the 'interesting ones' in particular to the underlying platform. While processing this broad spectrum of performance events one of the main considerations was to rank the performance events with respect to performance. The measure of performance is described differently in different scenarios but it is always related to time such as the number of instructions executed by the processor in a certain time or the number of packets sent or received in a particular time-frame over the network [10] are two different indicators of performance in two different scenarios. We consider the number of instructions passed through the 5-stage Reduced Instruction Set Computer (RISC) pipeline to serve as a key indicator of system performance (also explained in Section 3.1). Identifying the most frequent events with respect to performance would immediately determine the significant resource utilization behavior.

However, identifying significant performance events with respect to performance does not state how the resource utilization was performed. From hundreds of different performance events, any one can create thousands of data points per second of execution. This makes it hard to draw conclusions due to the given amount of data. Hence, the next challenge was to automatically analyze the acquired measurements. The captured data is complex for the reason that there is a logical hierarchy in the generation of different performance events coming from a particular hardware. It occurred as a result of multiple experiments that it is hard to find one single model that can describe the overall behavior of a performance event. Hence our interest moved toward simplification which is splitting the behavior into segments and providing the compact representation of dynamic-length segments based on statistical methods. The goal was not to quantify the magnitude of change but to have simple working sets based on the change in their data distribution.

To extend the analysis, the next goal was to automatically identify if per-

formance events can be grouped together based on their behavior. We consider PMU events can be related if they experience the changes in their behavior in a similar time fashion. The grouping criteria was not 'exactly the same' but 'somewhat similar'. Therefore by applying the change point detection method, the number of changes for each performance event in comparison can come differently. This mismatch in the number of changes restricts the use of traditional clustering algorithms because they need an identical feature as a classification rule. Besides, complex models are data-hungry for training themselves before making predictions and this deters the use of machine learning algorithms. Therefore, the challenge was to design and develop fuzzy matching criteria that can provide a symmetric feature of comparisons for finding the structure in the unlabeled data.

In summary, the thesis presents a mechanism to mitigate the hardware limitations through an automated approach that pinpoints the most relevant events for the applications' performance. The mechanism then enables us to understand the behavior through the compact representation of segments of performance events. Finally, an automated approach is presented to see if there are any groups of similar performance events based on changes in their behavior.

Thesis outline The thesis consists of two parts. Part I starts with presenting the overall research goals including the research approach and research method in Chapter 2. For a better reading experience, a technical background and related work are presented next in Chapter 3. Followed by the research methodology, results are presented in Chapter 4 to explain the contributions made through the published papers. Towards the end of Part I, a conclusion is presented along with prospective future work in Chapter 5.

Finally, the included papers are presented in Part II of the thesis.

Chapter 2

Research Overview

In this chapter, a brief description of the problem overview and research goals is provided. The chapter then continues to present the research method used to achieve the goals.

2.1 Problem Overview

Traditionally, a comprehensive view of systems performance is required for optimized and predictable usage of computer resources so we categorize the problems addressed through this thesis into the following three areas:

1. **Unstructured data and selection bias -** Specialized skill and training are prerequisites to a good analysis. So contextual understanding of both the tool and the examined architecture is required to draw accurate conclusions. Nevertheless, due to limited cross-platform compatibility, the current approaches are subject to selection bias. A more transparent data-driven solution is needed to automate the processes.

2. **Difficulty of analysis and visualization -** Data interpretation is hard due to the high volume of data generated by the complex system architectures. On one hand large volume of data is required for better analysis on the other hand sampling overhead serves as one of the grounds of statisti-

cal noise. All in all, analyzing complex data without visualization and analysis tools is challenging.

3. **Classification -** For complex data, underlying patterns and relationships are hard to reveal. Classification related to the change in behavior implicates a corresponding cause-and-effect relationship between different performance events such as resource dependence, concurrency, and resource contention. Simply applying traditional classification methods may not report the deep insights, we may need to tailor them to suit our specific requirements.

2.2 Research Goals

Performance events are important as they are tied to the resource utilization behavior of the applications. We consider the uncertainty gets higher in a shared resource environment if one does not know how the limited available resources are being utilized among different applications. Moreover, to encounter the challenges of the widely accepted practice of performing stand-alone tests based on a static list of performance events, an automated solution is a way forward to avoid human errors, expertise shortage, and manual effort. An evidence-based insight and understanding of the run-time behavior of the applications is indeed valuable. Therefore, we started with an overall goal of investigating the performance profile of applications. To achieve the overall goal we have divided it into three sub-goals.

Survivorship Bias - The Tale of the Forgotten Ones

Our first goal was to establish an automatic cross-platform mechanism that can capture the overall resource utilization behavior of the applications with respect to the performance metric. Rather than relying on selected performance events, the goal was to involve all of them, including the ones that usually do not pass the selection criteria due to engineers' knowledge and interest during the analysis. In short, the motivation was to include out-of-sight information while providing solutions and making decisions. Selection bias results in misleading insights and consequently leads to optimization misdirection, misguided resource allocation

and performance bottlenecks. An automated cross-platform approach is to reduce development costs, enhance collaboration, provide flexibility to adaption, reach a wider user audience and deliver consistent user experience.

RG1 – To establish an automated cross-platform mechanism to profile the performance of the applications.

Because Simple is Beautiful ...

Our next goal was to understand how the application performance and related system resource utilization evolve at runtime. The obstacle was finding a best-fit model for such a complex resource utilization behavior. So we aimed to countermeasure the complexity in the context of simplified data. Therefore, the goal was established to divide the behavior into dynamic-length segments such that the segments are being identified based on abrupt changes in data distribution.

RG2 – To establish an automated cross-platform mechanism for compact representation of performance events of applications.

What Makes it Related ...

We advanced the research by defining the next goal as identifying groups of performance events performing similarly. The data distribution may vary within each but if changes in their data distribution are occurring at similar points in time then we consider them related. So our focus was to investigate if they are related with respect to the time of change in their behavior.

RG3 – To automatically group up similar performance events of applications related to the time of change in their data distribution.

2.2.1 Research Method

In persuasion of an effective and practical research approach, empirical research inspired by design science [11] was conducted. The advantage is evidence-based insights to further explore the topic. We employed different kinds of applications for our experiments ranging from computationally heavy applications to memory-bound applications [12]. We have also tested a malicious application [13] that is known for the absurd exploitation of computer resources. The motive behind their selection is likelihood and significant use in industrial systems. Such applications can enormously impact the system performance due to their eager resource utilization demands. Table 2.1 maps the goals achieved in the corresponding papers.

Papers	RG1	RG2	RG3
Paper A	✓		
Paper B	✓	✓	
Paper C			✓

Table 2.1. Mapping of research goals into papers

In this thesis, Paper A proposes an approach to obstacle the architectural limitations of the underlying platform and aims to capture the overall cross-platform performance profile. The findings of Paper A endorsed the complexity of resource utilization behavior. So in the temptation of understanding the captured behaviour Paper B targets to model the system performance using statistical methods. In continuation to simplified behavior, the work continued in Paper C to explore the relationship between different performance events to see the signs of parallelism or influence on each other.

2.2.2 Research Approach

To conduct this research, an empirical approach inspired by design science is used. The overall research process is shown in Figure 2.1. We describe the steps as follows:

1. To achieve the overall goal, we formulated the problem to capture and

investigate the overall performance profile of applications.

2. Defined and outlined the mechanism to reach the research goals.
3. Designed and developed an artefact to provide the solution using statistical methods.
4. Preliminary results were obtained for further analysis using the developed artefact.
5. The preliminary results were evaluated to see whether the developed artefact solves the problem defined in step 1. There were two possible outcomes at this step:
   (a) The evaluation may give the ground to iterate back to step 2 to redefine the artefact.
   (b) The reasoning during the evaluation process can also advocate how the research can be expanded or if there is a need to go back to step 1 to reformulate the problem definition.

   The evaluation of preliminary results together with industrial partners determined when to stop this iterative process.
6. A proposed solution is presented as an overall contribution upon receiving satisfactory results.

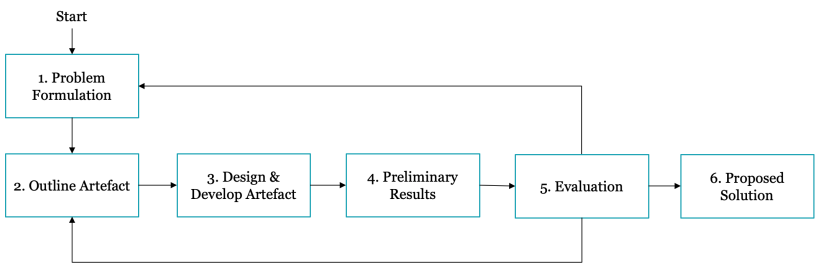


Figure 2.1. Research approach

Chapter 3

Background and Related Work

3.1 Performance

For improved performance of computers, the Reduced Instruction Set Computer (RISC) pipeline is a fundamental architectural feature of modern microprocessors [14]. And *Instruction* is the elementary unit a modern processor processes in one cycle, so the desire is to process more and more instructions in a given time. An instruction is marked retired as soon as it completes the 5-stage RISC pipeline. Concerning the current study, the more instructions retired in a certain time indicates higher performance.

The RISC pipeline is 5-stage process that allows concurrent execution of each stage, also shown in Figure 3.1. An instruction enters the first stage, called *Instruction Fetch (IF)*, in which the processor fetches the instruction from memory using the program counter (PC). After the current instruction is fetched, the process moves to the next stage called *Instruction Decode (ID)* where the processor determines what operations are to be performed. Here for a concurrent approach, the next instruction is fetched in the *IF* stage as soon as the current instruction is moved to *ID* stage. In continuation to the *ID* stage, now instruction is executed based on the determined operation in the *ID* stage and this stage is called *Execute (EX)*. During EX, if required, data is being written into or read from memory. Finally, after the execution, results are written back to the register at *Write Back (WB)* stage. The instruction is completed once it has passed the

write-back stage and is then marked as retired.

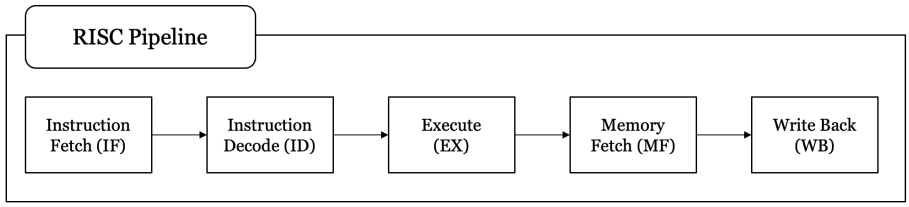


Figure 3.1. RISC pipeline for single instruction

3.2 Performance Monitoring Counters

The current state of modern computers enables us to precisely trace down the applications' resource usage at run-time. Modern computers have special built-in hardware in the Performance Monitoring Unit (PMU) in the form of direct memory access (registers). These hard-wired Model-Specific Registers (MSRs) can be configured to monitor the events occurring during a specific time interval in a system. An event is an observable activity, state or signal whose occurrence can be from different sources such as hardware, software, kernel etc [6]. The registers are generally named Performance Monitoring Counters (PMCs) and events are called PMU events or performance events [7, 6, 8].

PMCs are grouped into fixed-function counters and general-purpose counters, where fixed-function counters are hard coded and general-purpose counters can be programmed to monitor any kind of PMU event. The performance events are implemented by using processor-specific codes. These codes along with other attributes of events are provided by the vendor in JSON files i.e. arch event definition file. The number of available performance counters varies depending on the hardware architecture. A typical Intel processor contains 3 fixed-function and 4 general-purpose counters per PMU [7] and the IBM BlueGene supercomputer has 64 in total [15]. However, in a multi-core system, each core has its own set of PMCs. PMCs are not only available for CPUs but sometimes also for other components of the computer such as GPUs, network interface cards (NICs), network switches etc [16]. By using these PMCs, micro-architectural

performance events can be monitored in the processor pipeline, such as the branch predictor unit (BPU), internal memory events, off-core events, network resource utilization, network problems, etc even for the different components in parallel.

One advantage of using PMCs is the low overhead of data extraction [17] for performance events like branch instructions retired, mispredicted branches, cache hit/misses or floating point operations. There are tools for data extraction such as perf API which is a performance analysis tool and the official Linux profiler for both kernelspace and userspace. The perf API was originally developed for monitoring PMCs but evolved into a tool capable of tracing kernel activities too [6]. It uses processor-specific raw hardware descriptors for the performance events. These codes can then be translated into aliases (low-level human-readable event names) by using an event mapping table [18, 19]. The hardware vendor provides these hardware details in the form of JSON files (arch event definition files), as shown in Figure 6.2. In Linux, these JSON files can be located at *tools/perf/pmu-events/arch/foo*. The information is then used by Performance Application Programming Interface (PAPI) which aims to provide consistent and OS-independent access to PMCs.

PAPI was introduced as an abstraction layer to access the PMCs using the perf API interface. Over time PAPI has evolved into a component-based architecture, which can monitor data from multiple components like CPU, thermal sensors, network, virtual machines etc [16, 20]. PAPI extracts perf event codes and maps them into human-readable names based on the underlying platform to save users from low-level architectural details. These performance events are divided into two categories named *presets* and *native*. Presets are events that are common and consistent across the majority of the platforms (also called *architectural* [7]). However native events are specific to a given platform on which they are running (also called *non-architectural* [7]). Due to rapid advancements in technology and version changes, static solutions require frequent checks and updates which can directly influence the system performance. So using the 'native mask' non-architectural performance events can be extracted directly from the underlying hardware. The categorization of performance events and organization of profiling tools are also illustrated in Figure 3.2.

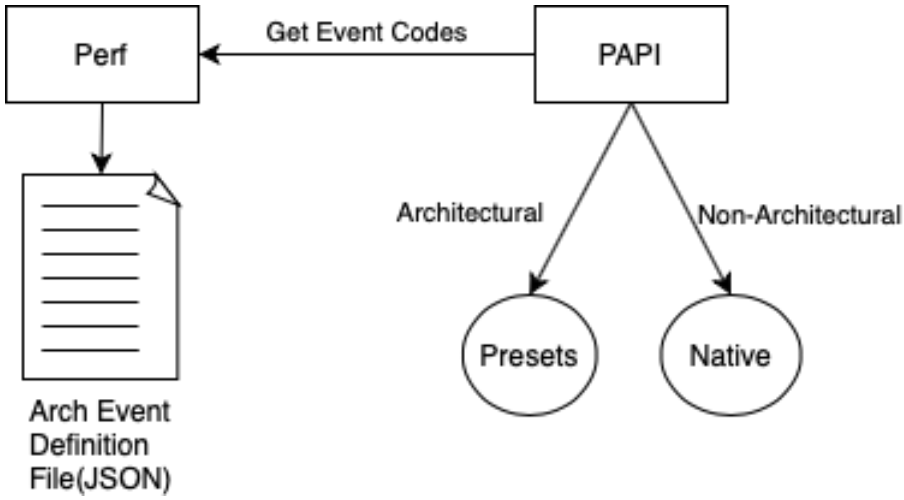


Figure 3.2. Illustration of perf API and PAPI in Linux Architecture

In short, when an event occurs it generates data that can further be utilized for statistical analysis as a metric or to generate an alert. These metrics are the result of evaluation or monitoring processes and can be used by technicians for system tuning and detection of faults. Events related to execution time, application memory-footprint size, memory latency, and error status can also present important insights.

3.3 Change Point Detection

Statistical methods are a conventional approach to analyze, interpret, and present huge amounts of data into a brief notation. Some of the common measures are standard deviation, mean and root mean square level to get valuable insights into data. However, there are numerous advanced methods also depending on the need and objective of analysis. Such as for determining segments in a data series, there are several methods like change point detection, cluster analysis, and time series segmentation.

Change point detection is one of the methods used for identifying distinct

points for partitioning a continuous data series. The method locates structural and distributional changes based on statistical methods like mean, standard deviation, and variance, also shown in Figure 3.3. The analysis can be parametric or non-parametric. A parametric analysis estimates by explicitly providing the location and/or the number of change points which is somewhat vulnerable to deviation [21]. On the other hand, the non-parametric analysis does not require a probability distribution assumption beforehand. These methods can be offline or online. Online methods use a subset of data series whereas offline methods use complete data series, from start to end, to make an analysis.

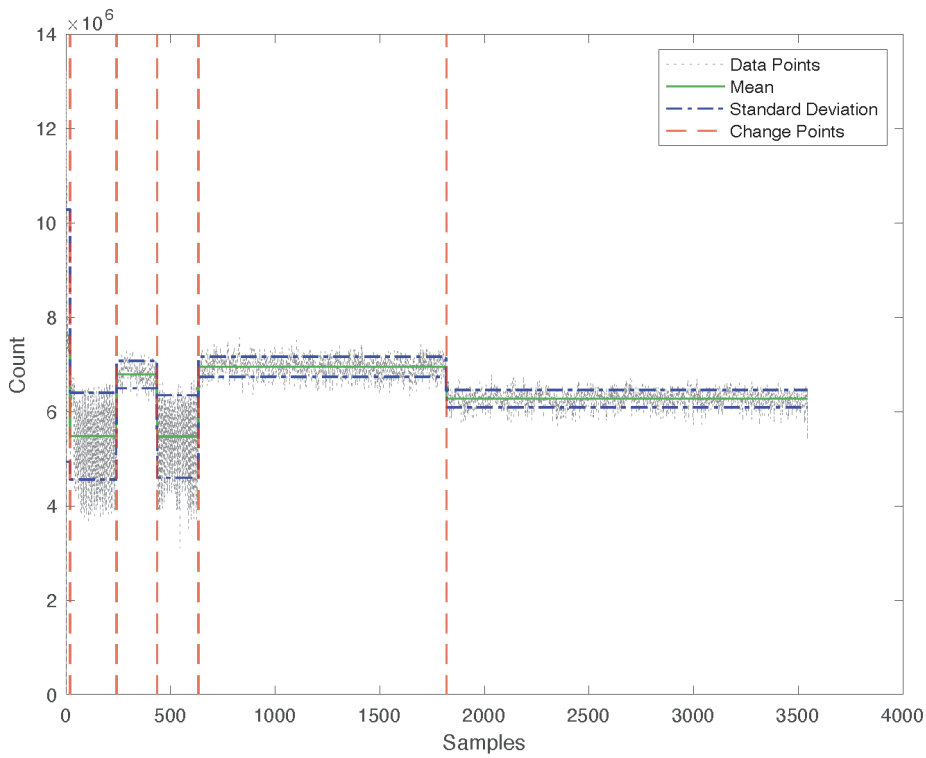


Figure 3.3. Change points detection for a measured performance event i.e., TLB_FLUSH

Some of the commonly used methods are likelihood ratio and Bayesian point

of view for single change point and multiple change point detection respectively. From a Bayesian point of view, it is possible to update the probability of the hypothesis with more data and a penalized contrast function [22]. The process is offline and the penalized contrast function starts with splitting the data series into two. Empirical estimation of statistical property (such as standard deviation, root mean square level, slope) is then calculated for each. Next, the sum of deviation from all the points in each part is calculated to see how much residual error still exists. The sum of aggregated deviations of each part gives a total residual error. This process is repeated until the final residual error is minimum [23]. Therefore, the Bayesian point befits the aim of our study. The result of the described process is also elaborated in Figure 3.3 for a measured performance event e.g., *TLB_FLUSH*.

It is also good to note some of the popular applications of change point detection are signal processing, genome, trend analysis, time series, intrusion detection, spam filtering, website tracking, quality control, step detection, edge detection, and anomaly detection.

3.4 Sequence Similarity - Similar Is Not Same

Sequence analysis or sequence similarity analysis is a popular method of identifying DNA similarity, a span of life trajectories & career and text similarity, alignment distances, document similarity and classification [24]. Some known methods are distance function (Chi-Squared, Euclidean), common attributes (Hamming distance, Longest common subsequence), Edit distance, Cosine similarity and Jaccard similarity. These methods are usually based on the measure of distance, order, position, time, duration and/or the number of repetitions [24]. Edit distance can be an appropriate choice if the aim is to quantify inequality. The method applies a weight for each edit function (insertion, deletion, substitution) until a sequence becomes identical to the other one. Cosine similarity is useful when similarity is not intended in terms of the size of the data. It can also be used in situations when the data sets are of different lengths and the orientation of the data is more important than the magnitude of the data [25]. Another method is Jaccard similarity which is a proximity measurement of shared properties i.e., size of intersection over the size of union [26].

All of these methods are based on an exact match of elements. However, in a classification problem, it is possible that some items are similar but not the same. Things are the same if they are identical to each other. Things can still be similar if they are not exactly the same. For example, if there are four sequences as below:

$$
\begin{aligned}
S_1 &= \{3, 5, 7, 1700\}, \\
S_2 &= \{3, 5, 7, 1700\}, \\
S_3 &= \{2, 5, 9, 1700\}, \\
S_4 &= \{3, 5, 1700\}
\end{aligned}
$$

We can see that sequence S_1 and sequence S_2 are the same because all of their elements are an exact match to each other. Yet S_3 and S_4 are similar to S_1 & S_2 because S_3 is slightly different for one element and S_3 is missing one value for an exact match.

3.5 Bounded Cost Function

In a matching principle, when the objects in comparison are not exactly the same the similarity is quantified with probability. There are many ways to compute the probability such as Binary step function, Linear functions, and Non-linear functions. The binary step function applies a static cost if a certain threshold is passed. The drawback is that it does not provide back-propagation. Linear functions are mean, variance, and covariance and they also do not offer back-propagation and the absence of one value can augment the cost of the others.

In comparison, there is a variety of non-linear cost functions such as Sigmoid, Hyperbolic tangent (Tanh), Rectified linear unit (ReLU), and Exponential linear unit (ELU) [27]. These functions have the advantage of proposing a smooth and bounded cost. For example, Sigmoid converts the number on a scale of 0 and 1 and gives the probability value as output, also shown in Figure 3.4. Its smooth scale gives the rate of change based on the gradient descent. The s-shaped

curve has one inflection point where the curve changes the shape from convex to concave. This point can serve as a decision boundary for classification.

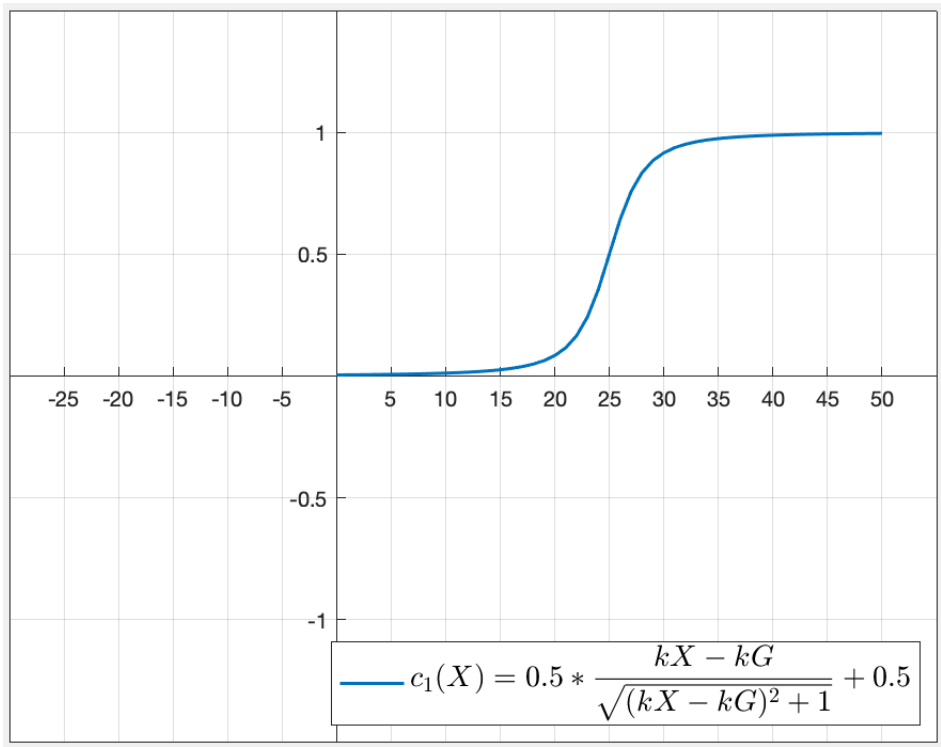


Figure 3.4. Graph of sigmoid function

The Sigmoid function is also important in artificial neural networks and logistic regression. Logistic regression is used to predict binary classification where Sigmoid plays the role of the activation function using its bounded scale. The bounded scale is reasonable to estimate the likelihood of probability which is why they are considered reliable to use with analysis algorithms for optimization purposes also. Therefore, we opt to use the non-linear Sigmoid function to calculate the cost to be applied while matching the sequences.

3.6 Related work

Here we present some of the related work in comparison to our research work and state-of-the-art. Most of the work in comparison is the one whose motivation comes mainly from the use of PMCs for various purposes. We also compare the ones who propose customized grouping and classification approaches.

The researchers [10] employed PMCs for the characterization of system performance and determined resource dependence of an application based on architectural events which are common across many platforms. One of the limitations of their study was to explicitly feed the performance events list for the characterization of the application. Their eventual focus was last-level caches only. Whereas our study focuses on all native events coming from the underlying platform to provide an automatic cross-platform solution.

Not only the cache but to explore other resources also there are studies that have used PMCs to estimate the power and bandwidth consumption [28, 17, 8] and to check the performance of applications in terms of CPU load and enhance quality of service by improving the performance [29]. Another study has used performance counters for the safety and security of the systems by proposing an attack mitigation model [30]. But as per our knowledge, these studies did not monitor all performance events of the platform they are running on. An interesting study performed by [15] on *Blue Gene/P*^{TM} was on a supercomputer to monitor the massive number of performance events (256 concurrent 64b counters). Although the capability to monitor performance was increased it is not very commonly available architecture across many Small and Medium Enterprises (SMEs) so the solution is not generally applicable.

There are situations when splitting the data set into segments becomes appropriate to provide a solution to the given problem. Several researchers have introduced various methods of segment detection. The well-known algorithms for change point detection are E-Agglomerative, Wild binary segmentation, Bayesian analysis of change points and Iterative robust detection of change points [31]. E-Agglomerative is a cluster-based approach to estimate change points depending on the goodness of fit [32]. The method is used to detect multiple change points within a data set. However, many of the methods require pre-screening to exclude the irrelevant points for improved accuracy which is

not the case with our proposed solution. We consider all the data points and focus on abrupt change after a stable behavior.

Yao considered multiple change points with the Bayesian point of view [33]. The Bayesian point of view is a form of statistical reasoning based on calculated probabilities to provide the best possible prediction. It is used when the inputs and information are not sufficient to determine the output. Yao also presented graph-based change point detection for high dimensional and non-euclidean data [34]. He studied the single-point case to estimate a change even when there is noise in data. The method can even estimate when the number of jumps is unknown and they are within defined bounds.

Another study [35] used randomly sampled basic block frequencies (sparse) without any dedicated hardware support and using PMCs. They propose Precise Event Based Sampling (PEBS) to reduce the run time overhead as one of the prime goals of their study. But it requires extensive normalization of data before processing.

To identify similarities and differences between multiple data sets some of the standard methods are least square and likelihood. However, it is not possible to directly apply the concepts due to inconsistencies in data and complicated requirements and conditions. There are other existing similarity approaches such as DNA similarity, Cosine similarity, Edit distance, and Jaccard index but they have preconditions like identical or different lengths, same data structure or exact match [24, 25, 26]. The way they compare is more strict and can be applied in absolute conditions. When it is not the case researchers like Fletcher and Islam [36] have used the Jaccard index for comparing the patterns coming from different techniques. Their proposed method converts each pattern into a single element which is also the commonality between their and our solution. However, the method to get a discrete value of similarity is different. Their method translates each pattern into an element of its own set whereas we compute the similarity based on element-wise weighted distance with respect to the lengths of the sequences. This is an additional strength of our proposed mechanism to handle the inconsistencies of data.

In situations involving limited data and diverse conditions for grouping, an approach has been applied by Koch, Zemel and Salakhutdinov [37] for one-shot image recognition where very limited or sometimes single example

is available to compare in supervised machine learning. They employed the sigmoid function in siamese convolutional neural networks to find the similarity between the final and hidden layer of the twin network. Their approach was to scale the absolute distance between 0 and 1 with the help of training parameters. Since their problem was binary classification so instead of utilizing real-value output the values from 0.5 to 1 were regarded as dissimilar. Whereas we use the resultant weighted cost as a probability of similarity. Moreover, their working sets were of the same length so one-to-one comparisons were directly possible which on the contrary was not a viable option for us. So we provide additional functionality to find the closest possible match with our holistic and intelligent approach.

Chapter 4

Research Results

4.1 Thesis Contributions

This section lists the contributions made through the goals achieved in this research, also shown in Figure 4.1.

- C_1: An automated cross-platform mechanism to capture the overall performance of the applications.

- C_2: An automated mechanism to identify the most relevant PMU events related to performance that describes what computer resources have been utilized most by the application under investigation.

- C_3: An automated approach to present a compact representation of complex performance events based on statistical methods

- C_4: An automated approach to group up similar performance events based on a measure of similarity

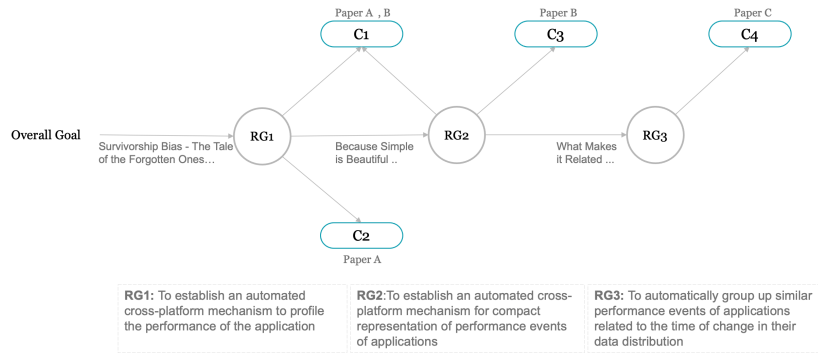


Figure 4.1. Mapping of research goals into corresponding contributions and papers

4.1.1 C1: An automated cross-platform mechanism to capture the overall performance of the applications

One of the core contributions made through Paper A [38] and Paper B [39] is the automated profiling of applications which has an enhanced ability to capture the overall behavior of applications on the platform where the application is running. The ability to capture the performance events native to the underlying computer architectures makes it consistent across different platforms. Besides, the automated solution is equally interesting for experts and non-experts in terms of ease of use, efficiency, competence, dynamicity, consistency and decision-making.

The consistent mechanism also ensures reduced human intervention and manual effort by characterizing all available performance events for the entire execution period of the application through re-run multiplexing. Whereas existing temporal multiplexing approaches are prone to blind spots due to which critical times may go unnoticed during event evaluation. Moreover, re-run multiplexing enables to characterize the short duration applications which can be problematic in the case of temporal multiplexing.

4.1.2 C2: An automated mechanism to identify the most relevant PMU events related to performance

The work in Paper A [39] continues to describe what computer resources have been utilized most by the application under investigation. The way the applications utilize computer resources indicates the likelihood of resource boundness, resource contention and security threats. Since different resources can be involved during the execution of an application the proposed mechanism is capable of reporting which hardware or software resources the application was utilizing most. Using statistical methods the mechanism ranks all characterized events with respect to performance to report the most relevant ones for the particular application under investigation.

4.1.3 C3: An automated approach to present a compact representation of complex performance events based on statistical methods

The work in Paper B [39] contributes not only to capture but also to understand the behavior of the applications. Interpretation of data generated as a result of complex resource utilization behavior was challenging to translate into one model. Various simplification and abstraction methods like polynomial and curve fitting techniques provided unsatisfactory results. Realizing the complexity, a sophisticated approach to simplify the working data set was determined i.e., to decompose the data into segments. There are many approaches for segmentation, current work emphasizes abrupt changes in data distribution as segmentation points. These changes can be identified using statistical methods like change point detection which enables to locate the significant moments in data shifts. Instead of investigating each point in time, considering the critical moments helps eliminate the impact of statistical noise caused by sampling errors.

4.1.4 C4: An automated approach to group up similar performance events based on a measure of similarity

A further contribution was made through Paper C [40] to group up similar performance events. There are different classification criteria to identify the

groups. However, due to quite a few limitations, applying a traditional clustering algorithm was not a viable solution in our case. The main challenge was to group up based on a 'not exactly same' but a 'somewhat similar' basis considering data is coming from complex, sensitive and rational resource utilization demands.

Another challenge was to deal with uneven lengths of series in comparison. Therefore, we have proposed a novel approach by tweaking the Sigmoid function and Jaccard index that considers the real value differences as cost while computing the weights of similarity. The methods can efficiently augment the existing clustering algorithms as a similarity function to compute the pairs of performance events. These pairs are then represented in a tree-structure i.e. dendrogram where the height of each linkage represents how different the performance events are from each other.

4.2 Included papers

A mapping of contributions to the corresponding papers is illustrated in Table 4.1.

	Paper A	**Paper B**	**Paper C**	
RG1	✓	✓		C1
RG1	✓			C2
RG2		✓		C3
RG3			✓	C4

Table 4.1. Mapping of contributions made through the papers for each research goal

4.2.1 Paper A

Title: Automatic Platform-Independent Monitoring and Ranking of Hardware Resource Utilization
Authors: Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jagemar
Status: Published in proceedings of 26th IEEE International Conference on

Emerging Technologies and Factory Automation (ETFA 2021)
Abstract: In this paper, we discuss a method for automatic monitoring of hardware and software events using performance monitoring counters. Computer applications are complex and utilize a broad spectra of the available hardware resources, where multiple performance counters can be of significant interest to understand. The number of performance counters that can be captured simultaneously is, however, small due to hardware limitations in most modern computers. We suggest a platform independent solution to automatically retrieve hardware events from an underlying architecture. Moreover, to mitigate the hardware limitations we propose a mechanism that pinpoints the most relevant performance counters for an application's performance. In our proposal, we utilize the Pearson's correlation coefficient to rank the most relevant performance counters and filter out those that are most relevant and ignore the rest.
My Contribution: Our industrial partner and co-author Marcus Jägemar has initiated the need for the problem to be solved. Following that, I was the main driver and author of this work. All authors have also contributed to the planning of the paper through productive and joint discussions. I have extended and implemented the characterization prototype provided by Jakob Danielsson. I have also written the first draft of the paper and all authors have contributed to improving the content with their valuable feedback.

4.2.2 Paper B

Title: Automatic Segmentation of Resource Utilization Data
Authors: Shamoona Imtiaz, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jägemar
Status: Published in proceedings of 1st IEEE Industrial Electronics Society Annual On-Line Conference (ONCON 2022)
Abstract: Industrial systems seek advancements to achieve required level of quality of service and efficient performance management. It is essential though to have better understanding of resource utilization behaviour of applications in execution. Even the expert engineers desire to envision dependencies and impact of one computer resource on the other. For such reasons it is advantageous to have fine illustration of resource utilization behaviour with reduced complexity. Simplified complexity is useful for the management of shared resources such

that an application with higher cache demand should not be scheduled together with other cache hungry application at the same time and same core. However, the performance monitoring data coming from hardware and software is huge but grouping of this data based on similar behaviour can display distinguishable execution phases. For benefits like these we opt to choose change point analysis method. By using this method our study determines an optimal threshold which can identify more or less same segments for other executions of same application and same event. Furthermore the study demonstrates a synopsis of resource utilization behaviour with local and compact statistical model.
My Contribution: I am the main driver and author of this work. All authors have also contributed to planning the paper through productive discussion. I have implemented the prototype and written the first draft of the paper and all authors have contributed to improving the content with their valuable feedback.

4.2.3 Paper C

Title: Automatic Clustering of Performance Events
Authors: Shamoona Imtiaz, Gabriele Capannini, Jan Carlson, Moris Behnam, Marcus Jagemar
Status: Published in proceedings of 28th Annual Conference of the IEEE Industrial Electronics Society (ETFA)
Abstract: Modern hardware and software are becoming increasingly complex due to advancements in digital and smart solutions. This is why industrial systems seek efficient use of resources to confront the challenges caused by the complex resource utilization demand. The demand and utilization of different resources show the particular execution behavior of the applications. One way to get this information is by monitoring performance events and understanding the relationship among them. However, manual analysis of this huge data is tedious and requires experts' knowledge. This paper focuses on automatically identifying the relationship between different performance events. Therefore, we analyze the data coming from the performance events and identify the points where their behavior changes. Two events are considered related if their values are changing at `approximately` the same time. We have used the Sigmoid function to compute a real-value similarity between two sets (representing two events). The resultant value of similarity is induced as a similarity or distance

metric in a traditional clustering algorithm. The proposed solution is applied to 6 different software applications that are widely used in industrial systems to show how different setups including the selection of cost functions can affect the results.
My Contribution: I was the main driver and author of this work. All co-authors have also contributed to the planning of the paper through productive and joint discussions. Together with Gabriele Capannini and Jan Carlson, we discussed different approaches for the proposed method that I have been implementing to evaluate their validity. Finally, the prototype provided by Gabriele Capannini was selected and then implemented by me. I have also written the first draft of the paper after which I and Gabriele Capannini have rewritten the methodology to its final version. All authors have also contributed to improving the content with their valuable feedback.

4.3 Publications not included in the thesis

4.3.1 Paper X

Title: Towards Automatic Application Fingerprinting Using Performance Monitoring Counters
Authors: Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jägemar
Status: Published in proceedings of 7th International Conference on the Engineering of Computer Based Systems (ECBS 2021)
My Contribution: I am the main driver and author of this work. All authors have also contributed to planning the poster paper through productive discussion. I have written the first draft of the paper and all authors have contributed to improving the content with their valuable feedback.

Chapter 5

Conclusion and Future Work

The main goal of this thesis is to understand the resource utilization behavior of the applications when running on a particular platform. Various performance events are available per platform, we aimed for an automated approach to providing better insights of application behavior. In system performance monitoring, CPU and memory are among the main sources of information [6] however the information can be collected from software, hardware and kernel as well. We have identified the challenges coming from hardware limitations, vendor specifications and lack of documentation while collecting and processing this information. The proposed mechanism does not require significant modifications to implement across different platforms since it unfolds the platform-specific list of available performance events to start its operations. Moreover, the proposed mechanism is resistant to selection bias which otherwise can be a reason to miss signs, clues and details of determining undesired state. These challenges were handled in the best possible way to establish a good foundation for comprehension and knowledge about application behavior. The proposed solution has also made an advancement to approach the aim of reduced investigation time and effort through an automated approach.

We have further explored and developed a solution to understand different phases in the behavior of each performance event. It was hard to present the data using one model so a simplified approach was determined i.e., segmentation. This enabled us to only consider the points in time where a change in resource

utilization demand is expected after a stable behavior. All these times can be considered during resource management to handle the parallel resource utilization demand from concurrent applications. This improves efficiency and saves from continuous monitoring. The proposed method does not require normalization which is a pre-condition for many methods. As a result of segmentation, the simplified subsets allow focused analysis, simplified visualization and deeper insights.

Finally, a contribution is made towards a relative picture of performance events with respect to changes in their behavior. It is hard to see a relationship between different performance events due to the vast variety of architectures and available performance events per platform. If the data is small then it still can be analyzed with extra effort but in case of the sheer volume of data, which is a result of complex behavior also, is time-consuming, error-prone and challenging. So a mechanism is proposed here that can compute the proximity of similarity between performance events by applying weighted real-value costs to relate different performance events. The automated mechanism reports groups of similar performance events with a decent accuracy based on concurrent changes in their behavior. The proposed solution can serve as a baseline feature to determine the relation and dependency between different resources.

5.1 Future Work

Solving complex performance problems usually requires holistic approaches. Finding an issue is not a problem, finding what matters the most at the time of decision is a challenge. The aim is to create a fingerprint of applications using its performance events. Having a fingerprint of an application can serve various purposes such as identification, detection or even decision-making.

In the future, we will extend the investigation for the segment-wise understanding of performance events. We are also interested in exploring how the segmentation can be performed with less number of measurements. The more the data the better examples are for comparison and change detection. One possible direction was to apply machine learning models but even the basic machine learning models are data-hungry to learn from the previously seen data. And when we are working on a very low level then it becomes a big overhead

to run these computationally heavy models. So the aim is to look for more sophisticated and lightweight statistical methods that can provide better insights even when there are not many or any examples to be compared.

Another immediate extension can be relating the trends in the data before and after the segmentation points to identify the impact between different resources. This way one can focus on the magnitude of the change for budget management which is another performance management problem that results in over-provision of the resources. Moreover, a future step is to provide a user-friendly publicly available tool to benefit from its capabilities. Making it open-source would allow improvements over time with the help of the wider community.

This also needs to be further explored what insights can be drawn to identify any relation at different resource levels. Currently, available documentations are poorly maintained that even the manual analysis using the performance event name is a problem. So a solution that can already relate the different data sets of performance events to their names, codes and details can supplement the documentation and analysis activities.

Finally, our aim is to present a model that can portray the best overall resource utilization behavior of the application.

Bibliography

[1] Marshall McLuhan. *The Gutenberg Galaxy: The Making of Typographic Man*. University of Toronto Press, 1962.

[2] Gabriel Balaban, Ivar Grytten, Knut Dagestad Rand, Lonneke Scheffer, and Geir Kjetil Sandve. Ten simple rules for quick and dirty scientific programming. *PLoS Computational Biology*, 17(3):223–231, 2021.

[3] Shirley Gregor, Leona Chandra Kruse, and Stefan Seidel. Research perspectives: the anatomy of a design principle. *Association for Information Systems*, 2020.

[4] Loic Brevault, Mathieu Balesdent, and Ali Hebbal. Multi-objective multidisciplinary design optimization approach for partially reusable launch vehicle design. *Journal of Spacecraft and Rockets*, 57(2):373–390, 2020.

[5] Vitaly Petrov, Thomas and, and Iwao Hosako. IEEE 802.15.3d: First Standardization Efforts for Sub-Terahertz Band Communications toward 6G. *IEEE Communications Magazine*, 58(11):28–33, 2020.

[6] Brendan Gregg. *Systems performance: enterprise and the cloud*. Pearson, 2nd edition, 2020.

[7] Intel. Intel® 64 and ia-32 architectures software developer's manual. Technical report, Intel, 2022.

[8] García-Martín Eva, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[9] Marc Mangel and Francisco J. Samaniego. Abraham wald's work on aircraft survivability. *Journal of the American Statistical Association*, 79(386):259–267, 1984.

[10] Jakob Danielsson, Tiberiu Seceleanu, Marcus Jägemar, Moris Behnam, and Mikael Sjödin. Resource dependency analysis in multi-core systems. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 87–94. IEEE, 2020.

[11] Paul Johannesson and Erik Perjons. *An introduction to design science*, volume 10. Springer Cham, 2014.

[12] Louis-Noël Pouchet. Polybench/c the polyhedral benchmark suite. `https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/#description`, 2023.

[13] IAIK. Meltdown. `https://github.com/IAIK/meltdown`, 2023.

[14] Aneesh Raveendran, Vinayak Baramu Patil, David Selvakumar, and Vivian Desalphine. A RISC-V instruction set processor-micro-architecture design and analysis. *International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA)*, pages 1–7, 2016.

[15] Valentina Salapura, Karthik Ganesan, Alan Gara, Sexton Gschwind, John James C., and Robert E. Walkup. Next-generation performance counters: Towards monitoring over thousand concurrent events. *ISPASS 2008-IEEE International Symposium on Performance Analysis of Systems and software*, 139-146:189–204, 2008.

[16] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with PAPI-C. In *Tools for High Performance Computing 2009*, pages 157–173. Springer, Berlin, Heidelberg, 2010.

[17] Stéphane Eranian. What can performance counters do for memory subsystem analysis? In *Proceedings of the 2008 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness: Held in Conjunction with the Thirteenth International Conference on Architectural Support for*

Programming Languages and Operating Systems (ASPLOS '08), MSPC '08, page 26–30, New York, NY, USA, 2008. Association for Computing Machinery.

[18] Linux Foundation. pmu-events. `https://github.com/torvalds/linux/tree/master/tools/perf/pmu-events`, 2023.

[19] Intel Corporation. perfmon. `https://github.com/intel/perfmon`, 2023.

[20] Matthew Johnson, McCraw Heike, Shirley Moore, Phil Mucci, John Nelson, Dan Terpstra, Vince Weaver, and Tushar Mohan. PAPI-V: Performance Monitoring for Virtual Machines. *41st International Conference on Parallel Processing Workshops*, pages 194–199, 2012.

[21] Changrang Zhou, Ronald van Nooijen, Alla Kolechkina, and Markus Hrachowitz. Comparative analysis of nonparametric change-point detectors commonly used in hydrology. *Hydrological sciences journal*, 64(14):1690–1710, 2019.

[22] Marc Lavielle. Using penalized contrasts for the change-point problem. *Signal processing*, 85(8):1501–1510, 2005.

[23] MathWorks. findchangepts - Find abrupt changes in signal, 2023.

[24] Matthias Studer and Gilbert Ritschard. What matters in differences between life trajectories: a comparative review of sequence dissimilarity measures. *Journal of the Royal Statistical Society: Series A (Statistics in Society), 179: 481-511,*, 179(2):481–511, 2016.

[25] Syed S. Ali Zaidi, Muhammad Moazam Fraz, Muhammad Shahzad, and Sharifullah Khan. A multiapproach generalized framework for automated solution suggestion of support tickets. In *International Journal of Intelligent Systems*, pages 3654–3681, 2022.

[26] Jiajie Peng, Jingyi Li, and Xuequn Shang. A learning-based method for drug-target interaction prediction based on feature representation learning

and deep neural network. In *BMC bioinformatics*, volume 21, pages 1–13, 2020.

[27] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task. In *rXiv preprint arXiv:1804.02763*, 2018.

[28] Rafia Inam, Mikael Sjödin, and Marcus Jägemar. Bandwidth measurement using performance counters for predictable multicore software. *IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–4, 2012.

[29] Marcus Jägemar. *Utilizing Hardware Monitoring to Improve the Quality of Service and Performance of Industrial Systems*. Doctoral dissertation, Mälardalen University, 2018.

[30] Alberto Carelli, Alessandro Vallero, and Stefano Di Carlo. Performance Monitor Counters: interplay between safety and security in complex Cyber-Physical Systems. *IEEE Transactions on Device and Materials Reliability 19*, pages 73–83, 2019.

[31] Shilpy Sharma, David A. Swayne, and Charlie Obimbo. Trend analysis and change point techniques: a survey. In *Energy, Ecology and Environment*, volume 1, pages 123–130, 2016.

[32] Hossein Alilou, Carolyn Oldham, Don McFarlane, and Matthew R. Hipsey. A structured framework to interpret hydro-climatic and water quality trends in Mediterranean climate zones. *Journal of Hydrology*, 614, 2022.

[33] Yi-Ching Yao. Estimating the number of change-points via schwarz'criterion. In *Statistics & Probability Letters*, volume 6, pages 181–189, 1988.

[34] Yi-Ching Yao and S. T. AU. Least-squares estimation of a step function. *The Indian Journal of Statistics, Series A*, pages 370–381, 1989.

[35] Andreas Sembrant, David Eklov, and Erik Hagersten. Efficient software-based online phase classification. In *2011 IEEE International Symposium on Workload Characterization (IISWC)*, pages 104–115. IEEE, 2011.

[36] Sam Fletcher and Md Zahidul Islam. Comparing sets of patterns with the jaccard index. *Australasian Journal of Information Systems*, 22, 2018.

[37] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. *ICML deep learning workshop*, 2(1), 2015.

[38] Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, and Marcus Jägemar. Automatic platform-independent monitoring and ranking of hardware resource utilization. In *26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2021.

[39] Shamoona Imtiaz, Moris Behnam, Gabriele Capannini, Jan Carlson, and Marcus Jägemar. Automatic segmentation of resource utilization data. In *1st IEEE Industrial Electronics Society Annual On-Line Conference (ONCON 2022*, pages 1–6. IEEE, 2022.

[40] Shamoona Imtiaz, Gabriele Capannini, Jan Carlson, Moris Behnam, and Marcus Jägemar. Automatic clustering of performance event. In *28th Annual Conference of the IEEE Industrial Electronics Society (ETFA 2023*, pages 1–8. IEEE, 2023.

II

Included Papers

Chapter 6

Paper A: Automatic Platform-Independent Monitoring and Ranking of Hardware Resource Utilization

Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jägemar
In the 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2021).

Abstract

In this paper, we discuss a method for automatic monitoring of hardware and software events using performance monitoring counters. Computer applications are complex and utilize a broad spectra of the available hardware resources, where multiple performance counters can be of significant interest to understand. The number of performance counters that can be captured simultaneously is, however, small due to hardware limitations in most modern computers. We suggest a platform independent solution to automatically retrieve hardware events from an underlying architecture. Moreover, to mitigate the hardware limitations we propose a mechanism that pinpoints the most relevant performance counters for an application's performance. In our proposal, we utilize the Pearson's correlation coefficient to rank the most relevant performance counters and filter out those that are most relevant and ignore the rest.

6.1 Introduction

Due to modern trends towards real-time data acquisition, inter-connectivity, data exchange and automation, *Industry 4.0* has revolutionised the industrial technology into cyber physical systems (CPS), Internet of things (IoT) and cloud computing. While bringing improved functioning, enhanced communication capabilities and shared services, this digital transformation has also put an increased pressure on engineers and system administrators. For them to keep such infrastructure functional, efficient, reliable and secure, it is more than ever required to conduct systematic health checks of computer systems and apply performance monitoring routines. A good knowledge of hardware resource demand and utilization by the hosted applications would facilitate the engineers, system administrators and auditors to ensure the Quality of Service (QoS) and security of IT infrastructure from undesired use.

The hardware resources required by an executing application may differ over the time. The demand could be for dedicated and/or shared resource(s) which is on discrete disposal, time scheduled [1] or managed through isolation techniques [2]. Observing resource utilization can reveal a distinctive behaviour of the application and can be used to tune the quality assurance process. Furthermore, in-depth analysis of performance monitoring data can ensure that system is performing as it is expected and can capture the execution profile of an application.

System performance Monitoring can be categorised into processor utilization, disk activity, memory usage and network usage. Modern computers have performance monitoring units (PMUs), responsible for monitoring the micro-architectural events. There are on-chip hard-wired special sets of registers known as performance monitoring counters (PMCs). The type and number of micro-architectural events are absolutely dependent on the underlying architecture and so is the number of PMCs. Regardless of the architectural variations from platform to platform there are events which are consistently available between many models, but this number is quite low and the terminology of event names are not identical across platforms. In these cases, the operator is bound to rely on information coming directly from vendor. Based on specific architectural knowledge, the PMCs can be configured to record hardware event metrics, but

the limited number of physical counters bound the number of events that can be monitored simultaneously.

PMUs are not only available for CPUs, but also for other components of the computer such as GPUs, network interface cards (NICs), network switches etc [3]. By using these PMCs, micro-architectural events can be monitored in the processor pipeline, such as the branch predictor unit (BPU), internal memory events, off-core events, network resource utilization, network problem etc even for the different components in parallel. The current state of modern computers enables us to precisely trace an applications' resource-usage at run-time. In this paper, we attempt to tackle the two following problems:

Application execution. An application typically displays an exceptionally complex execution trace and will utilize several resources simultaneously. Due to the complex execution trace, it is difficult to assess what resources are most relevant to the application's performance. Some applications display a performance that is closely tied to specific resources. We call occurrence *resource-dependency* and can be critical to understand when designing a system.

Performance Monitoring Counters. The number of available PMCs is limited in comparison to number of PMU events that can be observed at a time so that it is difficult to assess which events are important to monitor for a given application.

Other researchers have employed PMCs 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 static and pre-selected set of hardware events. Jägemar et al. [4] proposed a service associated with CPU scheduler for an improved QoS through performance monitoring counters' measurements. Danielsson et al. [2] used performance monitoring counters to identify resource dependence of application in a multi-core system.

In this paper, we continue the topic of automatic detection of an application's resource dependency, i.e., how much an application's performance depends on a specific resource. Our paper presents a new approach that monitors all available PMU events (both software and hardware) and builds resource-dependency profile for the applications. We presents a holistic approach to measure and rank what PMU events are most closely tied to an application's performance. Our contributions are as follows:

- A cross-platform method to monitor hardware resource usage by utilizing the Performance Monitoring Unit (PMU) for large sets of performance events.
- A measurement approach to distinguish what hardware- and software-resources have the highest impact on an application's performance for the large-scale performance counter event sets.

We start by providing a technical background to easily understand the technical scope and contribution made through this work in Section 8.2. Next, Section 8.3 presents our approach to achieve the goal of the study as well as giving a theoretical definition of our work. Our implementation details are described in Section 8.4 and the experimental setup in Section 6.5. We discuss our results in Section 8.5 and then present our conclusions in Section 8.7. Finally, we relate our work to the state-of-the-art in Section 8.6 and the anticipated future work concludes the paper in Section 6.9.

6.2 Background

In this section we describe the PMU and discuss the differences between counters and events. We also discuss application performance in a typical computer and how an application's performance is related to certain PMU events.

6.2.1 Performance Monitoring

The concept of performance varies for different applications depending on their primary objective. For example, in network applications, performance is usually measured in number of packets sent per second whereas image processing applications use the frames per second metric. These performance metrics can in-turn depend on other hardware-related metrics such as *utilization and saturation* for memory and CPU, *Operation rate and operation latency* for file systems, *disk utilization and response time* for disks, *throughput, connections, error, TCP re-transmits and TCP out-of-order packets* for networks [5].

Computers perform tasks on the basis of a sequence of instructions. In a classic Reduced Instruction Set Computer (RISC) pipeline one *Instruction* is

processed in one cycle, as shown in Figure 6.1. In here the instruction goes through stages from *Instruction Fetch* to *Write Back*. Modern computers imple-

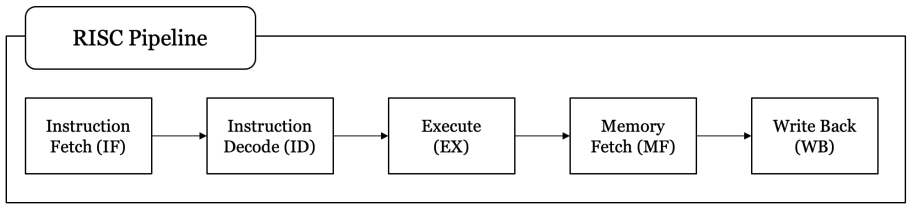


Figure 6.1. RISC Pipeline for a Single Instruction

ment this RISC pipeline for instruction-level parallelism to increase processor throughput. The classic RISC pipeline splits the execution of an instruction into five stages that are ideally able to work in parallel on different instructions. An instruction begins with entering the first stage *Instruction Fetch (IF)*. The instruction will then move to the second stage *Instruction Decode (ID)* once completing the IF stage, and another instruction will enter the IF stage. The instruction is completed once it has passed the write-back stage and is then marked as retired. It is, therefore, often preferable to have a high number of instructions retired for a given application since it indicates that the application is performing a lot of work. However, in case of application(s) executing *busy-wait loops*, *Instruction Retired* as a performance metric is not appropriate [6]. For example, in applications using sensors and actuators, actuators usually check the state of sensors which perform a busy-wait loop: this results in a large number of *Instructions Retired* even if the perceived performance still is low. Therefore, It is important to define the metric for performance based on use case.

6.2.2 Performance Monitoring Unit

Modern computers have special built-in hardware in the form of registers for performance monitoring. PMUs contain model specific registers (MSRs) those can be configured to monitor events. These hard-wired registers are also called performance monitoring counters (PMCs) [5] [7] [8]. Each core has its own set of counters. These counters count the number of occurrences of a certain event

during a specific time-interval.

PMC's are grouped into fixed-function counters and flexible-function counters, where fixed-function counters are hard coded and flexible-function counters can be programmed to monitor any type of event. The number of available performance counters varies depending on the hardware architecture, for instance a typical Intel processors contain 3 fixed-function and 4 flexible-function counters per PMU [7]. The event is an observable activity, state or signal whose occurrence can be from different sources such as hardware, software, kernel etc [5]. One advantage of using PMCs is negligible overhead of data extraction [9] for micro architectural events like branch instructions retired, mis-predicted branches, cache hits/misses or floating point operations. Usually the PMCs are implemented through processor specific codes. These codes along with other attributes of the events are provided by vendor(s) in JSON files which is arch event definition file.

When an event occurs it generates data that can further be utilized for statistical analysis as a metric or to generate an alert. These metrics are result of evaluation or monitoring processes and can be used by technicians for system tuning and detection of faults. Events such as execution-time, application memory-footprint size, memory-latency, and error status can also present important insights. Events those are present over majority of platforms are called architectural and events those are model-specific are called non-architectural events.

6.2.3 Perf and PAPI

Perf is a performance analysis tool and the official Linux profiler for both kernelspace and userspace. Perf was originally developed for the monitoring of PMCs but evolved into a tool capable of tracing kernel activities too [5]. Perf uses processor specific raw hardware descriptors for the PMC events. These codes can be translated into aliases (human readable event names) by using an event mapping table [10]. The hardware vendors provides these hardware-details in the form of JSON files (arch event definition files), as shown in Figure 6.2. In Linux, these JSON files can be located at *tools/perf/pmu-events/arch/<arch>*. The information is then used by PAPI which aims to provide consistent and OS independent access to PMCs.

Performance Application Programming Interface (PAPI) was introduced as an abstraction layer to access PMCs using the Perf interface. Over the time PAPI has evolved into component-based architecture, which can monitor data from multiple components like CPU, thermal sensors, Network etc [11] [12]. PAPI extracts perf events and maps them into human readable names based on the underlying platform to save users from low level architectural details.

These events are divided into two categories named *presets* and *native*. Presets are events which are common and consistent among majority of platforms (also called *architectural*). However native events are specific to a given platform on which they are running (also called *non-architectural*).

Due to rapid advancements in technology and version changes static solutions require frequent checks and updates which can directly influence the QoS in case of any delays and negligence. So the study is aimed to extract event list directly from underlying hardware such that the results are not dependent to out of date/static list of events at any point in time, as shown in Figure 6.2.

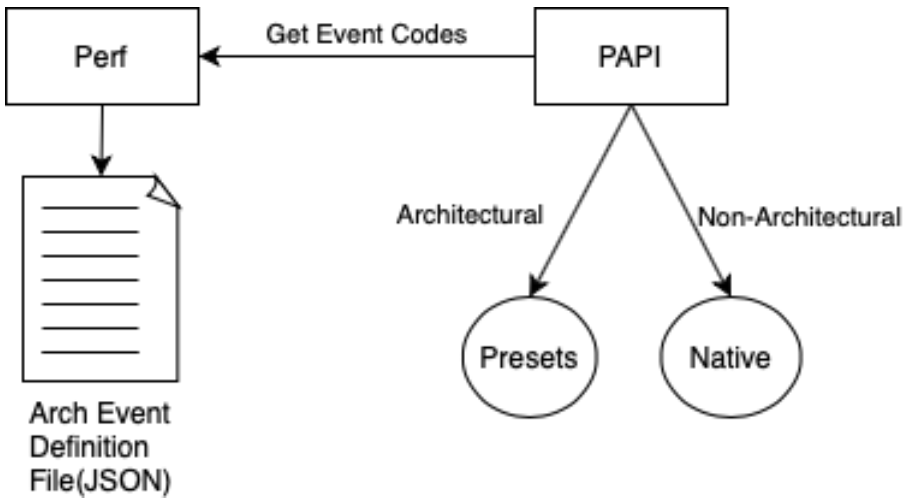


Figure 6.2. Illustration of Perf and PAPI in Linux Architecture

6.2.4 Multiplexing

Modern computers contain a vast number n of performance counter events. It is, however, not possible to simultaneously monitor n events due to high architectural and operational cost.

Multiplexing is a technique that can be used to monitor all events even if there are more events than counters. When number of events exceeds the core-internal PMCs then the technique is to configure core-external PMCs, if it is allowed. But if core-external PMCs cannot be utilized then time division multiplexing with core-internal PMCs is performed until full event coverage is done.

Perf automatically performs multiplexing by giving a fraction of time t_a to each event, in a round robin fashion [8]. This is done by switching frequency in Perf [13], and metrics are calculated usually at the rate of 100 to 1000 hz using formula:

$$C_T = \frac{C_R \times t_a}{t_e} \tag{6.1}$$

where C_R is counter value when an event got its turn to be monitored, t_e is the total time to monitor all the events and t_a is fraction of t_e when a particular event availed its turn to run. Here, C_T is an estimated value because it is not the count of an event throughout the execution period of an application.

6.3 Methodology

We summarize our ranking approach into three steps, listed as follows:

1. Event fetch – In this step, we execute an automatic traversal of n (all) available PMU events in the hardware architecture. Our fetch traversal step fetches the available events directly from the underlying platform.

2. Application characterization using Multiplexing– Here, we characterize an unknown application/process p using rerun multiplexing for n PMU events. As a result of multiplexing time-ordered series m_i of n PMU events are sampled and Pearson's correlation coefficient for each PMU event's time series is calculated as r_i.

3. Rank events – Finally, we sort the Pearson correlation values and highlight the R most important PMU events for application p.

6.3.1 Event fetch

Our method is focused on *native* events which is a main distinction from other studies in which proposed static solutions are dependant to a pre-compiled list of known events. We initialize the PAPI engine to traverse through all the available *components* (such as regular perf events and un-core perf events) on the current hardware, as shown in the Algorithm 1. We use the native event mask for event code generation which is the address of physical register where event details are stored.

```
1  initialize_PAPI();
2  setNativeEventMask();
3  num_components = getNumComponents();
4  component = 0;
5  while component ≤ num_components do
6      cmpinfo = getComponentInfo();
7      ENUM_flag = 0;
8      \* when ENUM_flag is set to 0 it iterates through all entries in
         descriptor file till the end of file *\
9      while (event_info = getEventInfo()) == TRUE do
10         \* Create Component wise event list *\
11         addEventsToCompEventList(event_info);
12         \* Move to next event *\
13     end
14     \* Move to next component *\
15     increment(component);
16 end
17 \* Create detailed list of native events for characterization of application
   *\
```

Algorithm 1: Get the PMU native event list

With *Enumeration* flag set to 0, we traverse through each object in event

description file. Function $getEventInfo()$ returns the information of next event available. This event information is then stored into a list. The event list is created per component so that we can distinguish that which event is configured on which component. When there are no more events, $getEventInfo()$ function returns 0 and loop exits. If there are more components available which are active then it moves to fetch events from that component. The process to get events is repeated in the same way for the next component. So in this way we iterate through the event list component by component and fetch n events details from each component.

6.3.2 Application Characterization using Multiplexing

Monitoring n events enables us to visualize the complete resource utilization profile of an application p. The obvious solution is multiplexing such as temporal multiplexing described in Section 6.2.4.

Temporal multiplexing is prone to blind spots. These blind spots are points in time when the event was not monitored and those times could be critical for an event evaluation. So we propose to run the application and monitor first subset sb of size no_PMCs events where $sb \subseteq n$ and re-run the application with next subset sb of size no_PMCs events and so on. In this way we can run the application for T_r times where T_r is total runs:

$$T_r = \lceil \frac{n}{no\_PMCs} \rceil \tag{6.2}$$

So, in our method of rerun multiplexing for complete coverage of events, we rerun the application quotient Quo times for no_PMCs events and then we run the application one last time to monitor remainder Rem events, also shown in Algorithm 2. Here Quo is $\frac{n}{no\_PMCs}$ and remainder Rem is $n \; mod \; no\_PMCs$.

Figure 6.3 shows the rerun multiplexing for core-internal PMCs in multiples of no_PMCs. If the total number of events n is not a multiple of no_PMCs then the difference is only for last iteration where Rem events are monitored. In each iteration, application is characterized by using the program designed by Danielsson et al. [2]. Characterization is performed with a sampling frequency

```
1  sb = no_PMCs;
2  Quo = n / no_PMCs;
3  Rem = n % no_PMCs;
4  while Quo ≥ 0 do
5      sb = get next no_PMCs from events(n);
6      characterizeApp(p, sb);
7      \* Store metrics and calculate Pearsonś Correlation coefficient*\
8      Quo = Quo - 1;
9  end
10 if Rem ≠ 0 then
11     sb = get next (Rem) from events(n);
12     characterizeApp(p, sb);
13     \* Store metrics and calculate Pearsonś Correlation coefficient *\
```

Algorithm 2: Re-run Multiplexing and Sampling of n events

freq for samples s over the total execution_time t_p of application as

$$freq = \frac{t_p}{s} \tag{6.3}$$

At the end of characterization each PMU event is sampled as time-ordered series, m_i. All series are then 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.

6.3.3 Ranking Events

We determine the most relevant events automatically by sorting them according to the correlation coefficient. The correlation between a specific event count and the number of instructions retired shows the application's resource dependence. In a way, the relation can simply be drawn by taking the difference of *total instructions retired* and *total count of* ith *PMU event* but Pearson's correlation coefficient can show the linear relationship between two variables.

Pearson's correlation coefficient is sensitive to outliers but in our case it is assumed to be natural even if the event data is not distributed evenly across the

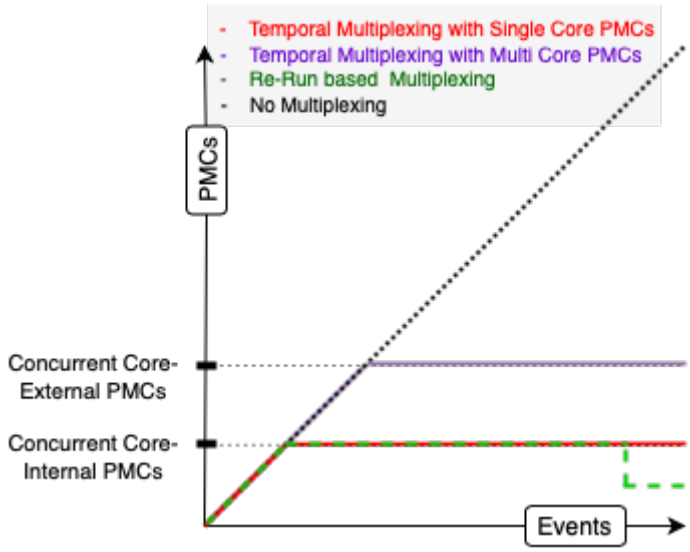


Figure 6.3. Illustration of rerun multiplexing in comparison to temporal multiplexing of Hardware Events for PMCs

timeline. Because it still shows there was any one or more points of times when this event has significant resource demand. However further research is required to know the exact points in time to profile the behaviour.

6.4 Implementation

We implement the proposed solution using Linux running *Ubuntu 4.13.0-21-generic* and *g++ 7.2.0*. PAPI library version 5.7.0.0 was used to iterate through all event codes.

As first step, we need to decide the sampling frequency $freq$, see Equation 6.3. Instead of applying fixed sampling frequency to every observed process/application, we calculate the sampling frequency based on the process's execution time t_p of p. The process is time stamped before and after the execution and difference between the two gives execution time t_p of p. For symmetric samples, we have opted to use a sampling size of $s = 1ms$ to calculate the $freq$

by using Equation 6.3. Since the t_p is calculated in micro seconds (μs), this sampling rate was experienced well to get enough number of samples as well as enough time to monitor the probe effect of an event.

To characterize an application for any number of events, a modification was made to the solution provided by Danielsson et al. [2] so that we can dynamically populate the event sets and feed those to PAPI engine and monitor n PMU events. In each iteration, a subset sb of events is monitored leaving the ones those were not able to attach. The reason a event cannot be attached is that the event is specified in the JSON event file, but not implemented on the actual hardware.

Once the characterization is done, Pearson's correlation coefficient of each PMU event's metrics are calculated. Their coefficients r_i are then sorted to rank the events such that higher the coefficient the higher the rank, with 0 being lowest and 1 being highest.

6.5 Experiments

We list some of the basic internal memory properties of our test computer in Table 6.1. Algorithm running on our experiment platform returns a total of 175 native events. We exemplify some of the events in Table 6.2.

Table 6.1. Hardware specifications Intel® Core™ i5 8250U

Feature	Hardware Component
Core	4xIntel® Core™ i5-8250U CPU (Kaby Lake) 1.6GHz
L_1-cache	32 KB 8-way set assoc. I-cache/core
	32 KB 8-way set assoc. D-cache/core
L_2-cache	256 KB 4-way set assoc. cache/core
L_3-cache	6 MB 12-way set assoc. Inter-core shared cache

Then we continue the experiment by choosing a test application, in our case it was a malware for side channel attacks known as *Meltdown* [14]. The reason to choose malware was they are naturally designed with a distinctive behaviour to achieve their purpose as compared to other general purpose applications. It is quite normal for a malware to stay unnoticed for a long time and trigger the hardware events suddenly in a specific time or environment. Due to their

Table 6.2. Some event from Native Event list

Event Code	Event Name	Description
0x4000006e	perf::LLC-STORES	Last level cache store accesses
0x40000073	perf::DTLB-LOADS	Data TLB load accesses
0x4000007d	perf::BRANCH-LOADS	Branch load accesses
0x40000089	INSTRUCTION_RETIRED	Number of instructions at retirement
0x400000ca	DSB2MITE_SWITCHES	Number of DSB to MITE switches
0x400000cc	FP_ARITH	Floatingpoint instructions retireds
0x400000d3	SW_PREFETCH	Software prefetches

unexpected behaviour it is more promising to catch any unusual activity in the event behaviour to visualize it as outlier or anomaly.

Running the community version of *Meltdown* variant on 4xIntel® Core™ i5-8250U CPU (Kaby Lake) 1.6 GHz a total of 175 native events from 2 components (perf event and perf event uncore) using PAPI. These non-architectural events are combination of available hardware and software events. For instance *ix86arch::BRANCH_INSTRUCTIONS_RETIRED* is a hardware event whereas *perf::PAGE-FAULTS* is a software event.

In Figure 6.4, we exemplify our characterization approach by utilizing the famous meltdown exploit as test application where $InstructionsRetired$ is used as performance metric. It is good to mention that these micro-architectural events are sensitive to the nature of application. Moreover, the selection of sampling frequency may significantly affect the results received.

We run the application 50 times and calculate the median of the Pearson's correlation coefficients. We list the events that displays the highest correlation coefficients in Table 6.3. The micro-architectural events occur at very low level and fast enough that any slight change in sample size affects the counter value significantly. Here, each run presents a high probability of counter value fluctuations, therefore, we rely on median of coefficients to present sound results.

Figure 6.4. Characterization of Application based on PMU events

Table 6.3. List of most relevant PMU events

Rank	Event Name	Coefficient
01	BR_INST_RETIRED	0.84
02	ix86arch::BRANCH_INSTRUCTIONS_RETIRED	0.79
03	perf::BRANCH-LOADS	0.52
04	perf::DTLB-LOADS	0.51
05	INSTRUCTION_RETIRED	0.51
06	perf::L1-DCACHE-LOADS	0.36
07	perf::BRANCHES	0.31
08	perf::BRANCH-INSTRUCTIONS	0.31
09	perf::PERF_COUNT_HW_BRANCH_INSTRUCTIONS	0.30
10	TLB_FLUSH	0.28
11	BR_MISP_RETIRED	0.25
12	BACLEARS	0.25
13	IDQ_UOPS_NOT_DELIVERED	0.22
14	ix86arch::MISPREDICTED_BRANCH_RETIRED	0.20
15	MOVE_ELIMINATION	0.19
16	perf::INSTRUCTIONS	0.18
17	perf::PERF_COUNT_HW_INSTRUCTIONS	0.18

6.6 Discussion

We perform measurements for all available hardware events on a computer and rank their relationships towards an application's performance using Pearson correlation coefficient. For the sake of this study we measure all hardware events regardless of their nature and redundancy. During event gathering, redundant event names were also observed whose one reason could be the presence of aliases such as $BR\_MISP\_RETIRED$ and $ix86arch :: MISPREDICTED\_BRANCH\_RETIRED$. They seem to be same as per available information but were listed under different event codes.

Though, temporal multiplexing can give a reasonable coverage of events but it is prone to blind spots. Not only the blind spots, counter value is also important to understand, which is an estimation based on the fraction of time it receives in round robin fashion. So for these two reasons there is high probability to miss the information as well as a chance of failure to observe the cascading effect of resource utilization at all. It is good to mention that cascading effects may only be observed through start-to-end or extended timeline processing. In contrast to temporal multiplexing our suggested rerun based multiplexing gives the complete picture of event behaviour for entire event range by utilizing all available PMCs.

As we do not parse the event description, it is required for once by the engineers to know the platform specific $InstructionsRetired$ event name from the acquired list of PMU events. This event name is then used as a benchmark to measure other PMU events during the characterization of application. It was also an option to take execution time as predictor but execution time may not be consistent all the times due to variable number of context switches in general purpose operating systems. And if the multiplexing is based on rerunning the application then it might not give us the same execution-time every time. So it is more reliable to take $InstructionsRetired$ as predictor for performance.

During each iteration, each event set tried to attach a subset of events but for some it was not possible such as for TLB prefetch misses, stalled cycles and blocking loads. The sampling for these events was not successful for $Meltdown$ as a test application but there is a good chance that those events can be monitored for some other application. Another reason could be that those events were

listed is arch event definition file (JSON) by vendors but were not available on actual hardware. Moreover, we left the costumed assignment and distribution of events to default between different cores.

Sampling frequency for the events that can be monitored was set to 1 ms which gave us around 50 samples each of test application as shown in Figure 6.4. Top most relevant events in Figure 6.4 shows that at the start there was high $InstructionsRetired$ rate and then a sudden drop. The number of $InstructionsRetired$ was quite consistent until just before the end of application where an exponential increase was observed for all relevant events. At first glance, it looked like an outlier but with a careful code analysis of test application the pattern of timeline was logical. In the start, higher activity was observed due to the exploit happening trying to crack down into kernel module from user space. Afterwards the utilization was smooth until the time just before the end of application's execution while reading the pre-fetch memory. The pattern did not show any distinction until it reached the point which according to the code is when the test malware application tried to remove its backtracks by calling a cleanup function. Due to this massive activity a spike is seen for high resource utilization.

Pearson's Correlation is normalized measurement of covariance to reflect the linear relationship between two variables. It is sensitive to outliers but in our case we assume outliers as legitimate points for evaluation. Even if the event data is not normally distributed across the timeline, it shows there was any one (or more) point of time when this event had significant resource demand. However further research can be done to know the exact points in time where the event leaves it marks and profile their behaviour.

Table 6.3 shows $BR\_INST\_RETIRED$ and $ix86arch :: BRANCH\_INSTRUCTIONS\_RETIRED$ as most relevant events which means that application was taking many branches. There was also high relevance to $perf :: DTLB\_LOADS$ which is a count of event when it reads from TLB. This observation actually brings the most interesting insight about the test application. A TLB load is lookup for actual physical memory address while using virtual memory. During this lookup, access privileges are also checked and if there is any permission violation it throws an exception. So the high relevance to this event indicates a distinctive behaviour of test application which we already know that it tries

to access kernel memory from user space and in that case there should be high exception rate. Such knowledge can further be used for categorisation and profiling of applications. Moreover, results showed that application is L1-Dcache bound too. So based on these event ranks engineers can automatically find out resource dependence during the execution of any application. Otherwise, it is based on operators' skills, experience and knowledge base only. The knowledge which comes from experience is valuable but it is good to keep in mind that human-driven approach is prone to mistakes, errors and insufficient skill set.

6.7 Conclusion

The study has successfully presented a solution to characterize any application p by sampling n base PMU events. The rerun based multiplexing enabled us to see the start-to-end event behaviour of event. Each sampled PMU event provided a time-ordered series, on which Pearson's correlation coefficients, r_i was calculated. Based on these correlations, ranking of events was performed to shortlist the most relevant PMU events for an application from the performance perspective. For experimental purposes a malware was tested for which our proposed service successfully listed the most relevant events. This knowledge can be further used for QoS, tuning and detection purposes. For instance, the results showed that $Meltdown$ was taking many branches and it was reading highly from TLB and $L1\_DCache$. Such kind of ranking of relevant events is indeed a useful tool for engineers to get better insights of health, performance and resource dependence of an application.

6.8 Relatedwork

The study is in continuation to the work done by Danielsson et al. [2]. The researchers determined the resource dependence of an application based on architectural events called PAPI *presets* which are common across many platforms. One of the limitations of their study was to explicitly feed the list of event names for the characterization of program with an eventual focus for last-level caches only. Whereas our study is focused for all native events to automatically extract from the underlying platform.

Rodrigues et al. [15] have used PMCs to dynamically estimate the power consumption by finding a minimal set of hardware events as a predictor. This study is restricted to a very small set of pre-selected hardware events based on human intelligence only. Also it lacks the statistical endorsement of selection of baseline events set. Moreover, the study used simulators instead of bare-metal environment which may jeopardize the accuracy of collected data. In contrast, our approach is aimed for bare-metal environment to capture as many as possible events by direct extraction from underlying platform.

There are other studies who have used PMCs to estimate the power and bandwidth consumption [16] [9] [17] and to check the performance of application in terms of CPU load [18]. Another study has used performance counters for safety and security of the systems by proposing an attack mitigation model [19]. But as per our knowledge, other studies did not automatically monitor all events regardless of which platform they are coming. Moreover, another interesting study was performed by [20] on *Blue Gene/P*^{TM} super computer to monitor massive number of PMU events (256 concurrent 64b counters). Although the capability to monitor performance was increased but it is not very commonly available architecture across many SMEs (Small and Medium Enterprises).

6.9 FutureWork

The study can be extended in many ways such as detection of faults, failure and malicious activity. Based on the hardware dependence a behavioural analysis of metrics can finger print any process. One of the biggest challenges is not only the low number of counters, but is to measure the events based on their nature such as configure the sampling frequency based on the nature of event to be monitored.

Occurrence of some events is not as frequent as others and for some the measurement cost at low frequency is too high. So a model built on top of event nature would improve the reliability of solution. Also, it would be interesting to test the measurement with other AI or statistical methods when the data distribution in non-linear.

Bibliography

[1] Marcus Jägemar, Andreas Ermedahl, Sigrid Eldh, and Moris Behnam. A scheduling architecture for enforcing quality of service in multi-process systems. In *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on*, pages 1–8. IEEE, 2017.

[2] Jakob Danielsson, Tiberiu Seceleanu, Marcus Jägemar, Moris Behnam, and Mikael Sjödin. Resource dependency analysis in multi-core systems. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 87–94. IEEE, 2020.

[3] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. *Collecting Performance Data with PAPI-C*. Springer, Berlin, Heidelberg, 2010.

[4] Marcus Jägemar, Andreas Ermedahl, Sigrid Eldh, and Moris Behnam. A scheduling architecture for enforcing quality of service in multi-process systems. *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2017.

[5] Brendan Gregg. *Systems performance: enterprise and the cloud*. Pearson, 2nd edition, 2020.

[6] Stijn Eyerman and Lieven Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, 2008.

[7] Intel. Intel® 64 and ia-32 architectures software developer's manual. Technical report, Intel, 2016.

[8] Andrzej Nowak and Georgios Bitzes. The overhead of profiling using PMU hardware counters. Technical Report CERN Openlab Report, CERN, 2014.

[9] Stéphane Eranian. What can performance counters do for memory subsystem analysis? In *Proceedings of the 2008 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness: Held in Conjunction with the Thirteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '08)*, MSPC '08, page 26–30, New York, NY, USA, 2008. Association for Computing Machinery.

[10] Linux Foundation. pmu-events, 2021.

[11] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with PAPI-C. In *Tools for High Performance Computing 2009*, pages 157–173. Springer, Berlin, Heidelberg, 2010.

[12] Matthew Johnson, McCraw Heike, Shirley Moore, Phil Mucci, John Nelson, Dan Terpstra, Vince Weaver, and Tushar Mohan. PAPI-V: Performance Monitoring for Virtual Machines. *41st International Conference on Parallel Processing Workshops*, pages 194–199, 2012.

[13] Stephane Eranian, Eric Gouriou, Tipp Moseley, and Willem Bruijn. Linux kernel profiling with perf. Technical report, Perf, 2015.

[14] Lipp Moritz, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. pages 973–990, 2018.

[15] Rance Rodrigues, Israel Koren, Annamalai Gracioli, and Sandip Kundu. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 882–886, 2013.

[16] Rafia Inam, Mikael Sjödin, and Marcus Jägemar. Bandwidth measurement using performance counters for predictable multicore software. *IEEE 17th*

International Conference on Emerging Technologies & Factory Automation (ETFA 2012), pages 1–4, 2012.

[17] Rodrigues, Rance and Annamalai, Arunachalam and Koren, Israel and Kundu, Sandip. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(12):882–886, 2013.

[18] Marcus Jägemar, Sigrid Eldh, Andreas Ermedahl, and Björn Lisper. Towards feedback-based generation of hardware characteristics. *7th International Workshop on Feedback Computing*, 2012.

[19] Alberto Carelli, Alessandro Vallero, and Stefano Di Carlo. Performance Monitor Counters: interplay between safety and security in complex Cyber-Physical Systems. *IEEE Transactions on Device and Materials Reliability 19*, pages 73–83, 2019.

[20] Valentina Salapura, Karthik Ganesan, Alan Gara, Sexton Gschwind, John James C., and Robert E. Walkup. Next-generation performance counters: Towards monitoring over thousand concurrent events. *ISPASS 2008-IEEE International Symposium on Performance Analysis of Systems and software*, 139-146:189–204, 2008.

Chapter 7

Paper B: Automatic Segmentation of Resource Utilization Data

Shamoona Imtiaz, Moris Behnam, Gabriele Capannini, Jan Carlson, Marcus Jägemar
In proceedings of 1st IEEE Industrial Electronics Society Annual On-Line Conference (ONCON 2022).

Abstract

Industrial systems seek advancements to achieve required level of quality of service and efficient performance management. It is essential though to have better understanding of resource utilization behaviour of applications in execution. Even the expert engineers desire to envision dependencies and impact of one computer resource on the other. For such reasons it is advantageous to have fine illustration of resource utilization behaviour with reduced complexity. Simplified complexity is useful for the management of shared resources such that an application with higher cache demand should not be scheduled together with other cache hungry application at the same time and same core. However, the performance monitoring data coming from hardware and software is huge but grouping of this data based on similar behaviour can display distinguishable execution phases. For benefits like these we opt to choose change point analysis method. By using this method our study determines an optimal threshold which can identify more or less same segments for other executions of same application and same event. Furthermore the study demonstrates a synopsis of resource utilization behaviour with local and compact statistical model.

7.1 Introduction

Many modern industrial systems require performance management to control machines, improve productivity and predict future problems. In performance management, it is critical to maintaining a satisfying service level to achieve business goals. These levels are highly susceptible to the resource utilization behavior of applications since the platform are different. However, different applications may have different resource utilization behavior during their execution. For example, an application is considered to be compute-bound if it mostly requires the processing unit. An application can be seen as I/O bound if it mainly utilizes I/O devices, and similarly, an application is memory bound if it has high memory utilization. For compute-bound applications, schedulers usually work well and keep the services aligned to Service Level Agreement (SLA) but in case of I/O bound applications, there is a possibility of some processes stressing others, especially in case of concurrent applications [1]. Therefore, it is valuable to know at what time a process demands more resources, such as cache memory, so that the engineers can separate the processes that has similar demand on same resources.

Such resource usage knowledge can be obtained through extensive resource utilization analysis of applications (running individually or in parallel with others). One way to get this information is through using Performance Monitoring Counters (PMCs). PMCs are special purpose, configurable, hard-wired registers available in the Performance Monitoring Unit (PMU) of modern processors to monitor PMU events [2]. These registers are used to count how many times a resource under observation is been utilized. Characterization of acquired measurements can indicate an application's behavior based on distribution of data over a period of time.

There are hundreds of different PMU events to select from, and any one typically can create thousands of data points per second of execution. Manual analysis of this wealth of data is very difficult and time-consuming. As shown in Figure 7.1 the utilization behaviour can be too rational, affinitive or hierarchical in nature that one single model is not able to represent such a complex data. However, a way to reduce the complexity is segmentation. Therefore, our contribution in this paper is to automatically analyse the acquired measurements

and determine the application behavior over time. More specifically the idea is to automatically identify segments in the data, similar to what Tukey formulated:

"At a low and practical level, what do we wish to do? We wish to separate the varieties into distinguishable groups, as often as we can without too frequently separating varieties which should stay together." [3]

Such distinguishable groups, which we call *segments* in our study, are often useful for practitioners, analysts, and engineers to intuitively visualize similar groups of data. These *segments* reflect a particular data distribution over a period of time. Several methods have been proposed to identify segments or groups such as clustering, segmentation, time series, change point detection [4], [5], [6] and basic block frequencies [7]. However, one of the widely used is change points detection which reports departure from the past norm. These change points split the behavior into segments based on abrupt change in the distribution and structure of data [6]. Each segment is segregated based on a model such as mean, median, standard deviation [5]. With such a statistical method it is possible to identify segments without any prior information both when sequences are independent or dependent of each other.

The case we are investigating here considers complete time series view and the objective is to provide more accurate estimation of change in time and magnitude. Therefore, the proposed solution is an offline method, as part of the manual analysis of applications, rather than online within a running system. This is also a common approach to evaluate applications before going into production or deployment. Such a solution can be standalone artifact or could be a part of a bigger intelligent tool. Overall, our work is towards automatically creating a resource utilization profile of an application. The main contributions presented in this paper are:

Segment detection: Our interest lies where the change is happening after a stable behaviour. In Figure 7.1(c), for *L1D_PEND_MISS*, around timed-sample 850 the count goes significantly up and keeps the level until before sample 1000. Such period we call a segment and we aim to automatically identify appropriate segments based on variance analysis.

Segment-wise statistical model: Once the potential segments are known, finding a local statistical model for each segment seems a viable solution for representing complex data series in a compact way using simple statistical

methods. The model provides information about how much resource utilization is expected during that interval of time. Thus, instead of permitting static over-provisioning for the entire execution period, resource allocation can be optimized to the segment lengths.

This paper begins with a technical background in Section 8.2 to provide relevant knowledge required to easily understand the contribution made through this work. Next, Section 8.3 explains the proposed approach to achieve the goal of the study. Section 8.4 extends the readers knowledge for implementation details and experimental setup. After successful implementation, results are being discussed in Section 8.5. We also relate our work to the state-of-the-art in Section 8.6. Finally the anticipated future work followed by the conclusion wraps up the paper in Section 8.7.

7.2 Background

We collect the data using Performance Monitoring Counters to observe hardware utilization of an application. The segments are then detected using Change Point Detection method. And finally a compact representation of each segment is provided using statistical methods. We present these concepts in Section 8.2.1, Section 8.2.2, Section 7.2.3 respectively.

7.2.1 Performance Monitoring Counters

The PMU is typically implemented as a set of registers programmed with a particular event to be counted. After a user specified time, the counted events can be read from the registers. These registers are configured to count events which is an observable activity, state, or signal coming from hardware, software, or kernel [8] such as Instructions Retired, Cache Hits, Cache Misses, CPU Clock Cycles. The name and number of events can vary on different platforms and different models [2]. The events which are common across other platforms are called architectural events (such as Instructions Retired, UnHalted Core Cycles) and events which are not consistent across various platforms are called non-architectural (such as L1D_PEND_MISS) [2]. The event-counting approach can be polling or sampling where polling means the arbitrary request for count

and sampling is an interrupt-based collection of event count [9]. An interrupt can either be generated based on time or when the counter exceeds a certain threshold. Our approach for data collection is interrupt-based timed sampling.

There are several tools available to acquire PMU measurements but it is possible to have variations in measurements depending on profiling tool, hardware type, starting time, reading technique, measurement level, noise etc. [9] but mostly they are good approximations.

7.2.2 Change Points Detection

Change point is a method of detecting structural and distributional changes based on statistical methods like mean, standard deviation, and variance. The analysis can be parametric or non-parametric. A parametric analysis estimates by explicitly providing the location and/or the number of change points which is somewhat vulnerable to deviation [10]. On the other hand, non-parametric analysis does not require a probability distribution assumption beforehand. These methods can be offline or online. Online methods use a subset of data series whereas offline methods use complete data series, from start to end, to make an analysis.

Some of the commonly used methods are likelihood ratio and Bayesian point of view for single change point and multiple change points detection respectively. From a Bayesian point of view, it is possible to update the probability of hypothesis with more data and a penalized contrast function [4]. The process is offline and the penalized contrast function starts with splitting the data series into two. An empirical estimation of statistical property (such as standard deviation, root mean square level, slope) is then calculated for each. Next, the sum of deviation from all the points in each part is calculated to see how much residual error exists. The Sum of aggregated deviations of each part gives a total residual error. This process is repeated until the final residual error is minimum [11]. Therefore, the Bayesian point befits the aim of our study.

Some of the popular applications of change points detection are signal processing, genome, trend analysis, time series, intrusion detection, spam filtering, website tracking, quality control, step detection, edge detection, and anomaly detection.

7.2.3 Statistical Methods

Statistical methods are a conventional approach to analyse, interpret, and present huge amounts of data into meaningful, brief notation. Statistics are valuable for engineers to identify working ranges, behavior, relations, level of significance and dispersion of data. Some of the common measures are standard deviation, mean and root mean square level. *Standard deviation* is the measure of spread, to show how much the data points are distant from the mean of the data set. A low *standard deviation* means that the data is closely clustered around the mean whereas a high *standard deviation* means that the data is dispersed over a wide range of values.

Since *standard deviation* is the square root of variance one might choose standard deviation over *variance* because it is a smaller unit, which in some cases is easier to work with. Also, it is less likely to get the impact of skewing. *Variance* treats all the numbers in the series in a same way regardless of whether they are positive or negative, which is an advantage when the direction of data is not important. A disadvantage of *variance* in case of larger outlying values is skewing so this is not necessarily a calculation that offers perfect accuracy [12].

Finally, to have a dimensionless analysis, we use *Coefficient of Variance*. It is a ratio of standard deviation to mean. Since it is percentage so the comparison between data of different units becomes coherent.

7.3 Proposed Solution

We have devised a method that is univariate because it involves one variable; the measurement of PMU event with respect to time. We start with presenting the definition of measurement approach, change point and segment.

Measurement Approach: For application, p, we define a set of PMU events, E, of size n. For each $e \in E$, a measurement series, r_m, is a series of L data points collected at frequency, f [13]. We run the test application x number of times so that multiple measurements for each $e \in E$ are acquired in $R_e = \{r_m : 1 \leq m \leq x\}$.

Change Point: For a measurement series, r_m, a change point, pts_j, is the point in time where the statistical model changes abruptly. A measurement series,

r_m, can have d number of change points such that $pts(r_m) = \{pts_j : 1 \leq j \leq d\}$.

Segment: Given a set $pts(r_m)$ of d values, we can split the series of L points in r_m into a partition of $d+1$ segments defined as $S(pts(r_m)) = \{[1, pts_1], (pts_1, pts_2], , ..., (pts_d, L]\}$.

The total number of segments may vary depending on the size and behavior of p. If no change point is detected then whole series is denoted by one segment. The number of change points are always one less than the total number of segments.

Applying these concepts, we propose a solution consisting of following steps:

1. Segment Detection – In this step, our method identifies a threshold for which root mean square error becomes persistently low. This threshold is considered optimal threshold and can be used as model threshold for any measurement of same PMU event of same application.

2. Segment-wise Statistical Model – Next, we find local model of each segment in terms of standard deviation and mean for a given segment length.

7.3.1 Segment Detection

Initially, our method determines the individual working threshold for each measurement so that an optimal threshold can be derived which can work for any measurement of a PMU event of a application. Thus we present three-step segment detection as:

1. **Compute Primary Threshold:** For each measurement, we compute the threshold for which the residual error is persistently low. We start with loading R_e measurements for x number of runs of an application at step *1*, as shown in Algorithm 32. Then through step *2* till *12*, by using different values as threshold from 1 to $maxThresh$ we determine where the residual error starts to increase. During this process we compute change points with threshold t of current iteration j at step *4*. Resultant threshold and residual error are initialized during first iteration at step *5*

but for later iterations through steps *8* to *12* we seek to identify threshold where residual error becomes consistently low. For example, if residual error was low at threshold 2 and same residual error was received at threshold 3 then it means the method can sustain low residual error up till threshold 3 therefore primary threshold should be 3 in such case. This was an important consideration during the experiment to nicely stop the detection process and report the primary threshold for PMU event e with decent accuracy.

2. **Compute Optimal Thresh:** Next, the method computes optimal threshold from all the primary thresholds at step *13*. The subroutine receives series of thresholds & residual errors and computes optimal threshold based on median of corresponding residual errors of primary thresholds from each measurement. Therefore first the median is computed at step *19* then matching residual error is identified through steps *21* to *24*. If the matching residual error is found then we take the corresponding threshold into $thresh$ otherwise we find a residual error closer to median of primary residual errors into $thresh$, as shown from step *22* to *23*. The subroutine then returns optimal threshold into th at step *13* as optimal threshold for a PMU event of an application.

3. **Compute Segmenfts:** Finally, we compute segments for each measurement from step *14* to *17* and report change points and residual error for a PMU event e.

This three step process can be repeated for any or each of the PMU event. Therefore above described method is illustrated as a method to determine optimal threshold th which can detect d number of change points. These change points eventually provides the number and length of segments as describe in the definition of *segment* in Section 8.3.

7.3.2 Segment-wise Statistical Model

Once the segments with decent accuracy are detected a compact illustrations of resource utilization behaviour of each segment is presented using statistical methods. A PMU event having zero or one segment shows no variability to

```
1  r contains x measurements in R_e
2  for j = 1 to x do
3      for t = 1 to maxThresh do
4          ⟨resError, pts⟩ = findchangepts(r[j], t)
           /* tr contains primary threshold for one
              measurement of PMU event e                    */
5          if t == 1 then
6              tr[j] = t
7              re[j] = resError
8          end
9          if resError == lastResError then
10             tr[j] = t
11             re[j] = resError
12             break
13         end
14         lastResError = resError
15     end
16 end
   /* th contains optimal threshold for all
      measurements of PMU event e                           */
17 th = computeOptimalThresh(tr, re)
   /* Find segments using optimal threshold                 */
18 for j = 1 to x do
19     ⟨resError, pts⟩ = findchangepts(r[j],th)
20     S[j].pts = pts
21     S[j].resError = resError
22 end
   /* Find optimal threshold                                */
23 function computeOptimalThresh(t, re)
24     med = median of re
25     thresh = 0
26     for j = 1 to length(re) do
27         if (re[j] == med) or (re[j] < med and re[j+1] > med) then
28             thresh = t[j]
29         end
30     end
31     return thresh
32 end
```

Algorithm 3: Find segments for PMU event e

model so such PMU event is not profiled. Also a PMU event with too many segments is also pruned away because it means the behaviour is too arbitrary or inconsistent to profile.

7.4 Implementation and Experiments

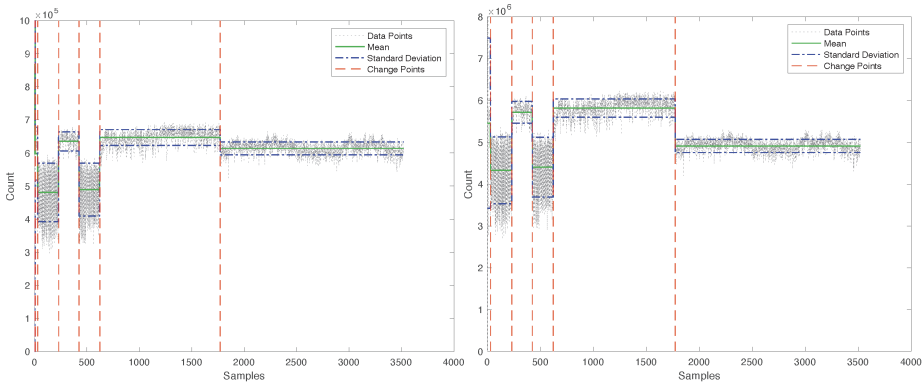
We implement the proposed solution using *PAPI library version 5.7.0.0* for the sampling of PMU events with 5 milliseconds frequency. The number of samples may go different depending on the execution time of the application to profile. Then, *findchangepts()* function in *Matlab version R2021* is used to find the segments. The measure of distinction to compute these segments is root mean square level. Evaluation of results is performed with the help of Coefficient of Variance.

Test Application: For the experiment we opt to chose *2x2 matrix multiplication* of *PolyBench* bench-marking tool, known for kernel instrumentation as a test application. The motive behind its selection is significant use of matrices in image recognition software. Such applications can enormously impact the system performance due to their eager resource utilization demands. The execution period of the test application is around 22 seconds which gives thousands of number of samples based on 5 millisecond frequency.

Measurements: The same test application was characterised 20 times for its complete execution period. For each execution period measurements were acquired through re-run based multiplexing of 4 PMCs available in 4xIntel® Core™ i5-8250U CPU (Kaby Lake) 1.6GHz using the solution provided by Imtiaz et al. [13]. The test application running on our experiment platform returned in total 172 native PMU events.

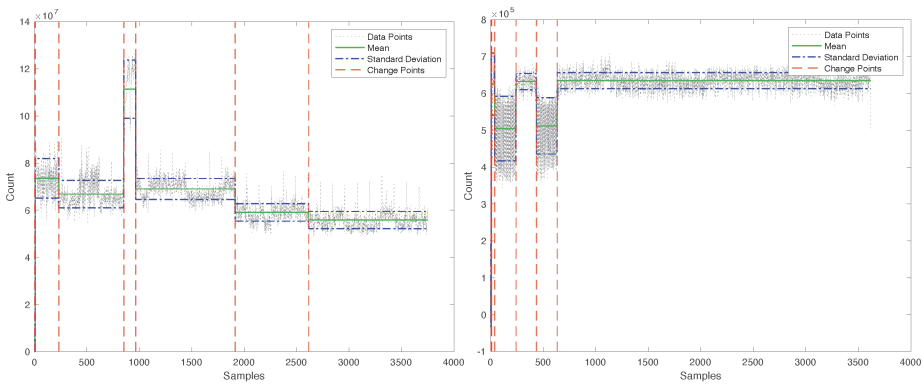
Results: The PMU event with zero, one or more than 20 change point(s) was pruned away as explained in Section 7.3.2. These bounds can be re-adjusted based on the total execution time of process and number of samples. As a result total 53 PMU events were identified with distinct pattern based on a statistical model. Lastly, we exemplify some of the PMU events with their segments in Figure 7.1.

At the end of experiment we evaluate the results to learn if an optimal threshold can find segments with low residual error for any measurement of



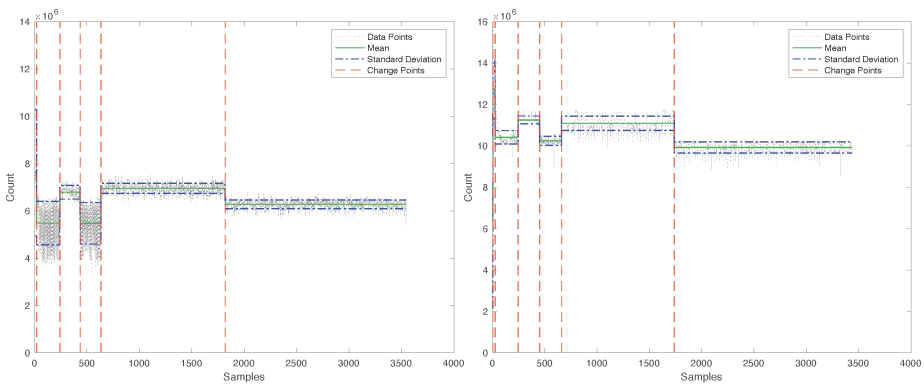
(a) Segments for Branch Instructions Retired

(b) Segments for Instructions Retired



(c) Segments for L1D_PEND_MISS

(d) Segments for L1D



(e) Segments for TLB_FLUSH

(f) Segments for perf-CPU-CYCLES

Figure 7.1. Simulation results for the network.

a same PMU event. Therefore the variance of residual error is examined and validated by calculating coefficient of variance (CoV). Since variance could be a big number depending on unit of data set so for the readability sake we prefer to express percentage. Therefore, coefficient of variance (CoV) is a reasonable choice which is defined as the ratio of the standard deviation (σ) to mean (μ) such that $CoV = 100 * \sqrt{\frac{\sigma}{\mu}}$.

Table 7.1 shows resultant CoV for some of the PMU events. We also present maximum residual error received by applying the proposed method in Table 7.1.

Table 7.1. Accuracy of segment detection method

PMU Event	**Coefficient of Variance (%)**	**Maximum Residual Error**
Branch Instructions Retired	2.63	1.0130e+05
Instructions Retired	2.66	1.1767e+05
L1D_PEND_MISS	1.79	1.3675e+05
L1D	1.68	1.0144e+05
TLB_FLUSH_DATA	1.98	1.1829e+05
perf-CPU-CYCLES	4.11	1.2375e+05

7.5 Discussion

The proposed dynamic approach automatically detects number and location of segments based on root means square level. The method does not need to know the number of change points as an input parameter to find segments. This sequential method takes the complete data series into account to be able to iteratively investigate and adjust key points until the residual error becomes minimum. The results in Figure 7.1 shows samples on x-axis and resource utilization count on y-axis. Vertically segmented series shows where there is a change in resource utilization behaviour. For instance PMU event *Branch Instructions Retired* shows how many branch instructions were completed when the event was sampled. Figure 7.1(a) shows how the trend is changing from one segment to other i.e. going up for segment starting around 250 to 450 and then it goes down during the next segment and then again it goes up and so on.

Also Table 7.1 for *Branch Instructions Retired* show CoV is only 2.63% and maximum recorded residual error from actual points is 1.0130e+05 which is quite nice accuracy for segmentation.

Change point is similar to outlier with a slight difference i.e in change point there is a time step into a new model (such as a change in mean value) whereas in the case of an outlier there is a significant time step out of a single model [5]. This we can see in data distribution of *TLB_FLUSH* in Figure 7.1(e). In segment starting around 600 and ending around 1850, time steps out of a single model are ignored as outliers, and the time series does not split into a new segment for each outlier. Therefore this segmentation approach is independent of pre-screening, pruning, or normalization of given data.

In Figure 7.1(d), last segment of *L1D* shows consistent higher L1D cache utilization starting roughly from sample 600 to 3700. This knowledge can be useful in the case of hyper-threading which allows to run more than one thread on each core. Applications running on a hyperthreaded CPU utilize two hardware threads that share the same physical processor and L1 cache so running them in parallel with higher L1 cache utilization may cause L1 congestion. Cache congestion can lead to bad or unpredictable application performance. Therefore, such knowledge can also be used by other automated tools responsible for decision-making or can be a standalone tool for critical analyses. For instance, a scheduler or container's orchestrator can consider these points to estimate the resource utilization during these segments.

Furthermore, in Figure 7.1(f) for *perf_CPU_CYCLES*, segment starting roughly from 250 to 490 shows high CPU cycle count which means the application was more active during this time interval and utilized more computational resources. The hype in the CPU cycles can be related to dynamic frequency scaling which allows the microprocessors to adjust the CPU frequency on the fly depending on the actual needs for power management. The segments with such hypes can then be useful if an application may require underclocking or overclocking.

7.6 Related Work

To identify similarities and differences between multiple data sets some of the standard methods are least square and likelihood. The algorithms for change point detection are E-Agglomerative, Wild Binary Segmentation, Bayesian analysis of change points and Iterative Robust detection of change points [6]. E-Agglomerative is cluster based approach to estimate change points depending on the goodness of fit [14]. The method is used to detect multiple change points within a data set. However, many of the methods require pre-screening to exclude the irrelevant points to obtain an improved accuracy which is not the case with proposed solution.

Multiple change points was also considered by Yao [15] where Bayesian point of view was involved which is a form of statistical reasoning based on calculated probabilities to provide best possible prediction. Bayesian point of view is used when the inputs and information is not sufficient to determine the output. Yao also presented graph based change point detection for high dimensional and non-euclidean data [16]. He took single-point case to estimate change even when there is noise in data. The method can even estimate when number of jumps are unknown and they are within defined bounds.

Another study used randomly sampled basic block frequencies (sparse) without any dedicated hardware support. They propose Precise Event Based sampling (PEBS) to reduce run time overhead as one of the prime goals of their study [7]. But it require extensive normalization of data before processing.

7.7 Conclusion and Future Work

The study has successfully presented how a change in resource utilization behavior can be automatically identified by using penalty-based function from a Bayesian point of view. The Bayesian approach iterates until the change in statistical function has a minimum residual error. This study has shown an improved automated approach to determine the empirical threshold that can provide segments without prior knowledge of the number of change points. With this method, the total number and location of segments is reported with a low segmentation cost. Such knowledge can be a component of performance

management system and can save from exceeding resource capacities. Moreover, when the data is small and solitary then differences can be visible to the human eye but when the data is huge, complex and continuous then a manual analysis can not help benefit the process management. Therefore, a boxed representation of each segment can be further used during performance management, QoS, tuning, and detection purposes.

Lastly, we keep working on extending the method into a forked activity which can then be used for reliable decision making purposes. One of the extension can be providing these segments details to orchestrator which can consider the resource utilization demand while scheduling the containers.

Bibliography

[1] Marcus Jägemar, Andreas Ermedahl, Sigrid Eldh, and Moris Behnam. A scheduling architecture for enforcing quality of service in multi-process systems. *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2017.

[2] Intel. Intel® 64 and ia-32 architectures software developer's manual. Technical report, Intel, 2022.

[3] John W. M Tukey. Comparing individual means in the analysis of variance. In *Biometrics*, pages 99–114. JSTOR Arts and Sciences II Scopus, 1949.

[4] Marc Lavielle. Using penalized contrasts for the change-point problem. *Signal processing*, 85(8):1501–1510, 2005.

[5] ArcGIS Pro. How change point detection works, 2022.

[6] Shilpy Sharma, David A. Swayne, and Charlie Obimbo. Trend analysis and change point techniques: a survey. In *Energy, Ecology and Environment*, volume 1, pages 123–130, 2016.

[7] Andreas Sembrant, David Eklov, and Erik Hagersten. Efficient software-based online phase classification. In *2011 IEEE International Symposium on Workload Characterization (IISWC)*, pages 104–115. IEEE, 2011.

[8] Brendan Gregg. *Systems performance: enterprise and the cloud*. Pearson, 2nd edition, 2020.

[9] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. SoK: The challenges, pitfalls, and perils of using hardware performance counters for security. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 20–38. IEEE, 2019.

[10] Yao Wang, Chunguo Wu1, Zhaohua Ji, Binghong Wang, and Yanchun Liang. Non-parametric change-point method for differential gene expression detection. *PloS one*, 6(5):e20060–e20060, 2011.

[11] MathWorks. findchangepts - Find abrupt changes in signal, 2023.

[12] Indeed. What Is Variance? Definition And How To Calculate It, 2022.

[13] Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, and Marcus Jägemar. Automatic platform-independent monitoring and ranking of hardware resource utilization. In *26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2021.

[14] David S. Matteson and Nicholas A. James. A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, 109(505):334–345, 2012.

[15] Yi-Ching Yao. Estimating the number of change-points via schwarz'criterion. In *Statistics & Probability Letters*, volume 6, pages 181–189, 1988.

[16] Yi-Ching Yao and S. T. AU. Least-squares estimation of a step function. *The Indian Journal of Statistics, Series A*, pages 370–381, 1989.

Chapter 8

Paper C: Automatic Clustering of Performance Events

Shamoona Imtiaz, Gabriele Capannini, Jan Carlson, Moris Behnam, Marcus Jägemar
In 28th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2023).

Abstract

Modern hardware and software are becoming increasingly complex due to advancements in digital and smart solutions. This is why industrial systems seek efficient use of resources to confront the challenges caused by the complex resource utilization demand. The demand and utilization of different resources show the particular execution behavior of the applications. One way to get this information is by monitoring performance events and understanding the relationship among them. However, manual analysis of this huge data is tedious and requires experts' knowledge. This paper focuses on automatically identifying the relationship between different performance events. Therefore, we analyze the data coming from the performance events and identify the points where their behavior changes. Two events are considered related if their values are changing at `approximately` the same time. We have used the Sigmoid function to compute a real-value similarity between two sets (representing two events). The resultant value of similarity is induced as a similarity or distance metric in a traditional clustering algorithm. The proposed solution is applied to 6 different software applications that are widely used in industrial systems to show how different setups including the selection of cost functions can affect the results.

8.1 Introduction

Computers are subject to hardware resource management in a fashion similar to cognitive load management in humans. Consider a technician decommissioning hardware from a data center while enjoying the music with headphones on. Meanwhile, a sudden alarm urges immediate attention. Listening to the music while handling the alarm does not amuse anymore, a vigilant response of the technician to handle the situation is imperative. This is cognitive load and switching off the music is required to manage the case. The same applies to machines, resource management is the amount of information a machine can load and process at one time. A system can possibly be efficient and competitive by improving its software and hardware utilization behavior. In general, it needs to be considered critically which applications are running in the environment and how they affect the system performance. This is why deploying industrial applications requires extensive resource utilization analysis in terms of computational cost, memory requirements, load balancing, scheduling and Quality of Service (QoS). Performance events such as *L1_Cache_Misses, Branch_Instruction_retired, Instructions_Retired, Page_faults, TLB_Misses* demonstrate the resource utilization of an application on a platform [1]. The knowledge gained is useful for software developers, system integrators and performance engineers in terms of load & efficiency, responsiveness and resource utilization analysis respectively.

One way to monitor performance events is the use of Performance Monitoring Counters (PMCs), available in the Performance Monitoring Unit (PMU) of the modern computer. These hard-wired special purpose registers are used to record the events that occurred by the utilization of software and low-level hardware resources such as cache, memory, processor, translation lookaside buffers (TLB) and data bus. On the one hand, it is possible that multiple performance events are related to the same hardware resource and on the other hand one performance event may cause the generation of another performance event. Moreover, in a shared resource environment the impact, relations and dependency between different performance events are more significant than apparent. A shared cache can really destroy the performance and increasing the cores to serve the simultaneous hardware utilization demand of all the applications running in parallel is not an efficient use of resources. However, the

interpretation of such a complex execution behavior is indeed hard to visualize without an intelligent tool. Moreover, tools to investigate which performance events are related are not openly available if there exists any. Consequently, the probability of missing the signs and clues of parallelism, dependencies and relationship between different resources gets higher.

Nonetheless, a problem cannot be solved without solving the cause and not every performance problem can be solved in one scenario. Hence, the focus of the study is to identify the existence of a relationship between different performance events with respect to time. However, the captured resource utilization behavior is sensitive to the sampling rate and the platform where the application is running. For such reasons, finding one single model for complex behavioral data is itself a challenge. One way to reduce the complexity of the working set is using segmentation [2]. Segmentation divides the longer sequence into smaller parts (segments) based on applied criteria. The positions where the sequence is chopped down are called segmentation points. So rather than point-to-point comparison, identifying relationships based on the segmentation points is more appropriate to circumvent the sampling bias and different loads of the execution environment. It also gives the advantage of breaking down the rational and affinitive behavior based on the abrupt changes in the data distribution [3, 4]. Therefore, instead of quantifying the magnitude of change for the sake of similarity, we consider performance events to be related who experience a change from their previous norm in a similar time fashion.

Groups of related data sets can be determined statistically through clustering analysis. However, clustering analysis is not limited to one approach. Regardless of the constitutional basis of a clustering algorithm (such as distance, density, or connectivity) [5] all clustering algorithms require at least one homogeneous feature for grouping. Having change points as a feature of similarity hinders the use of off-the-shelf clustering algorithms since the number of changes in each performance event is unlikely to be the same. That being the case, we have instrumented the traditional Hierarchical Clustering algorithm with a customized cost function to handle the inconsistency of the data sets. This work builds upon our previous work on automated performance monitoring [6] and segmentation of resource utilization data [7], addressing clustering of performance data based on the identified segments. The provision of such knowledge helps engineers to

make interpretations based on their interests. They can target a particular group of performance events rather than multiplexing the hundreds of performance events, especially when the number of performance monitoring counters is limited per platform. Therefore, our main contributions are:

Similarity based on segmentation points: Our first contribution is to identify not the same but somewhat similar performance events based on the segmentation points. Based on that we calculate the proximity of similarity for the ordered sequences of numbers. The length of ordered sequences in comparison can be different.

Clustering based on customized distance function: Our second contribution is to compute the pairwise matrix using our similarity function. Such that the matrix can be utilized by the traditional clustering algorithms to identify the groups of similar performance events.

Briefly, we start our study by presenting a technical background in Section 8.2 for the readers to easily understand the contribution made through this work. Next, we present our proposed solution in Section 8.3 describing the approach used to achieve the goal of the study. The implementation details and the experimental setup is then outlined in Section 8.4. Following the implementation details, results are discussed in Section 8.5 to extend the reader's knowledge. The state-of-the-art and related work to our study is then presented in Section 8.6. Finally, the anticipated future work followed by the conclusion concludes the paper in Section 8.7.

8.2 Background

We use the Performance Monitoring Counters (PMCs) to capture the resource utilization data of the applications. The segmentation points are identified using the change point detection method. Having these working sets ready for analysis similarity is measured using different cost functions. The weights of these costs are then considered during the grouping of performance events.

8.2.1 Performance Monitoring Counters

The processor is one of the main information sources when observing activities in computing devices. A fixed number of registers are available to record the low-level performance information at the CPU cycle level [8]. The performance information is an observable activity, state or signal coming from hardware, software, or kernel. This observable activity is called an event and there can be hundreds of such events that are been generated by the application when it is running on a platform. In comparison to a large number of available events, only a limited number of registers are available per platform. These registers are called Performance Monitoring Counters (PMCs) [1]. Though the number of available PMCs is processor-specific, they are always significantly less than the available performance events [1, 9, 10]. Therefore, multiplexing is required when monitoring a high number of performance events. A non-standardized event naming and numbering scheme between hardware architectures further complicates the portability of low-level performance monitoring tools. However, a significant effort has been made by the cross-platform tool called Performance Application Programming Interface (PAPI) to standardize the performance events names [11]. PAPI also gives the ability to collect all platform-specific performance events called native events. In this study, we use PAPI to collect performance monitoring data.

8.2.2 Change Point Detection

Change point detection is a well-known statistical method for locating changes in terms of mean, variance, or standard deviation of a data set. The offline method requires complete data series beforehand and applies a penalized contrast function to locate the positions of abrupt changes in the entire data distribution. The iterative method continues to repeat until the aggregate deviation based on empirical estimate becomes minimum, called residual error [3, 4]. The parametric function allows different attributes for trend analysis such as the number of changes, the statistical method to be applied, the threshold for minimum residual error improvement, and the minimum distance between change points.

Some of the popular applications of change point detection are signal processing, genome, trend analysis, time series, intrusion detection, spam filtering,

website tracking, quality control, step detection, edge detection, and anomaly detection. We use change point detection to identify the change in trend such that the change in the behavior of one performance event leads towards an impact on another performance event's behavior.

8.2.3 Sequence Similarity - Similar Is Not Same

Sequence analysis or sequence similarity analysis is a popular method of identifying DNA similarity, a span of life trajectories & career and text similarity, alignment distances, document similarity and classification [12]. Some of the known methods are distance function (Chi-Squared, Euclidean), common attributes (Hamming Distance, Longest Common Subsequence), edit distance, cosine similarity and Jaccard similarity. These methods are usually based on the measure of distance, order, position, time, duration and/or the number of repetitions [12]. Edit Distance can be an appropriate choice if the aim is to quantify the inequality. It applies a weight for each edit function (insertion, deletion, substitution) until a sequence becomes identical to the other one. Cosine similarity is useful when similarity is not intended in terms of the size of the data. It can also be used in situations when the data sets are of different lengths and the orientation of the data is more important than the magnitude of the data [13]. The Jaccard similarity is a proximity measurement of shared properties i.e., size of intersection over the size of union [14].

All of these methods are based on an exact match of elements. However, in a classification problem, it is possible that some items are similar but not the same. Things are the same if they are identical to each other. Things can still be similar if they are not exactly the same. For example, if there are four sequences as below:

$$
\begin{aligned}
S_1 &= \{3, 5, 7, 1700\}, \\
S_2 &= \{3, 5, 7, 1700\}, \\
S_3 &= \{2, 5, 9, 1700\}, \\
S_4 &= \{3, 5, 1700\}
\end{aligned}
$$

We can see, the sequence S_1 and sequence S_2 are the same because all of their elements are an exact match to each other yet S_3 and S_4 are similar to S_1

and S_2. The matching criteria of this study is similarity.

8.2.4 Bounded Cost Function

In a matching principle, the similarity is quantified with probability when the objects in comparison are not exactly the same. There are many ways to compute the probability such as the Binary step function, Linear functions, and Non-linear functions. The binary step function applies a static cost if a certain threshold is passed. The drawback is that it does not provide back-propagation. Linear functions are mean, variance, and covariance and they also do not offer back-propagation and the absence of one value can augment the cost of the others. In comparison, there is a variety of non-linear cost functions such as Sigmoid, Hyperbolic Tangent (Tanh), Rectified Linear Unit (ReLU), and Exponential Linear Unit (ELU) [15]. These functions have the advantage to propose a smooth and bounded cost. For example, Sigmoid converts the number on a scale of 0 and 1 and gives the probability value as output. Its smooth scale gives the rate of change based on the gradient descent. The bounded scale is good to estimate the likelihood of probability which is why they are considered reliable to use with analysis algorithms for optimization purposes. The Sigmoid function is also important in logistic regression. Logistic regression is used to predict binary classification where Sigmoid plays the role of the activation function. Therefore, we use the non-linear Sigmoid function to calculate a decent cost to be applied while matching the sequences.

8.2.5 Clustering Analysis

Clustering analysis is a classification technique for the grouping of objects with respect to a predefined matching rule. A rule can be distance, density, location, similarity, or any. There are various clustering algorithms: Hierarchical, Density-Based Spatial Clustering of Applications with Noise (DBSCAN), K-means and Nearest Neighbor [16].

In Hierarchical clustering, a multilevel hierarchy of clusters is formed such that each cluster is found based on a similarity function between the data points. First of all pairwise distance based on similarity is computed and then clusters of objects are created using close proximity. The iterative method keeps growing

the binary tree into a larger cluster based on the pairs. However, not to forget, Hierarchical clustering does not perform well when the data set is too large. DBSCAN is also a distance-based clustering method and can work not only with traditional distance measurements like Euclidean, Manhattan and Hamming distance but also with customized distance functions. A customized distance function can then be used during the computation of the distance matrix for making decisions. Both Hierarchical and DBSCAN, do not need to know the number of clusters to find beforehand therefore this makes them good candidates for clustering for this study.

8.3 Proposed Solution

We propose a two-step method that identifies the groups of similar performance events based on our customized similarity measurement.

8.3.1 Similarity Detection

This section firstly describes the measurement approach, segmentation points, and sequence used in this study and proposed in [7]. Then our method for evaluating the similarity of two given sets of segmentation points is introduced. To this end, we defined our own similarity function which is inspired by the Jaccard similarity coefficient [17]. While the original definition is based on counting the number of identical elements occurring in two input sets, we aim to have a more flexible definition of matching between a pair of elements. This has been achieved by exploiting a pairwise cost function mapping the difference between two elements into the range $[0, 1]$ where zero denotes a perfect matching, i.e., the elements are the same. By means of the proposed cost functions, we defined our similarity measure and, then, populated a similarity matrix that acted as input for the chosen clustering algorithm i.e., Hierarchical clustering.

Measurement approach For a given application, p, we define a set of performance events, E, of size n. For each $e \in E$, a measurement series, m_i, of L_i data points is collected at frequency, f [6]. As a result, we get n measurement

series for the application p such that $M(p) = \{m_i : 1 \leq i \leq n\}$ where m_i is a time ordered series of i-th performance event.

Segmentation points and Sequences Segmentation points are the points where abrupt changes in measurement are coming and the sequence is an ordered list of successive numeric elements. Using the statistical change point detection method, the segmentation points are identified through the mechanism devised in [7]. Thereby an ordered sequence $pts(m_i)$ of d_i number of segmentation points for each m_i is detected such that $|pts(m_i)| \in (1, d_i]$. Each element in $pts(m_i)$ corresponds to a perceived change in trend after a stable behavior and $|pts(m_i)|$ can be different for each m_i.

Cost Functions Let p_1 and p_2 a pair of points belonging to $pts(m_1)$ and $pts(m_2)$, respectively. A cost function returns a higher value when p_1 and p_2 are more distant. The cost function is required to be defined in $[0, +\infty)$ and return a value in $[0, 1]$ where values near 1 denote a higher difference between p_1 and p_2.

As candidate cost functions we defined three monotone non-decreasing functions depicted in Figure 8.1 where $x = |p_1 - p_2|$ and defined as:

$$c_1(x) = 0.5 \cdot \frac{k_1 x - k_1 g_1}{\sqrt{(k_1 x - k_1 g_1)^2 + 1}} + 0.5 \tag{8.1}$$

$$c_2(x) = \frac{k_2 x}{\sqrt{(k_2 x)^2 + 1}} \tag{8.2}$$

$$c_3(x) = \min(1, (g_3 x)^2) \tag{8.3}$$

Here, any g parameter (i.e., g_1 and g_3) is a similarity threshold defining the dividing point between "similar points" (if $x < g$) and "different points" (if $x > g$) while k (i.e., k_1 and k_2) is a positive factor used to flatten the functions thus for tuning the degree of uncertainty in defining the similarity value returned. Parameter g selection can be challenging in terms of level of strictness therefore the g is been used in three different scenarios through Equations 8.1, 8.2 and 8.3 as firm, rigorous and strict, respectively. Whereas parameter k is tweaked to create the 'S' shape curve of the Sigmoid function. In more detail:

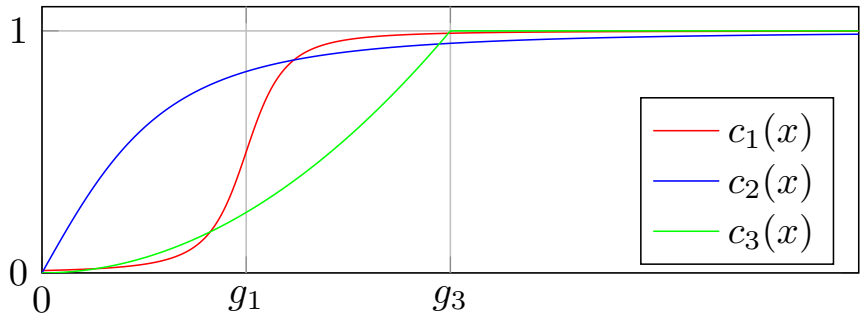


Figure 8.1. Plot of the cost functions proposed with the thresholds g_1 and g_3.

- The first cost function c_1, Equation 8.1, is a non-linear Sigmoid function having the parameter g_1 as similarity threshold that denotes the point where the function becomes concave, as shown in Figure 8.1. The function also requires the parameter k_1 which denotes how fast the function approaches the extremes (i.e, 0 and 1) while getting away from g_1.
- The second cost function c_2, Equation 8.2, corresponds to the concave part of another non-linear Sigmoid function. Here there is no parameter defining the similarity threshold (i.e, $g = 0$), like in c_1, just the factor k_2 for flattening the function, which is to set how quickly it reaches the upper extreme while getting away from zero. This function considers the absolute distance between p_1 and p_2 regardless of the similarity threshold so the cost is calculated at one step function hence resulting in a higher aggregated cost.
- Our last cost function c_3, Equation 8.3, is a quadratic function where a factor g_3 defines the point of maximum distance between p_1 and p_2 after which the two compared points returns the maximum cost.

In the following, one of the cost functions is applied at a time to test the accuracy of each.

Similarity Function Let $A, B \in pts(m_i)$ be two sequences of ordered numbers to be compared. Our similarity function is inspired by Jaccard similar-

ity [17] defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{8.4}$$

For our similarity function, rather than using the number of elements that are present in both sets, $|A \cap B|$, we calculate the matching degree between the two sets by means of a given cost function c.

In particular, let $|B| \leq |A|$, we assess $|A \cap B|$ as the result of summing $1 - c(|b - a^\star|)$ for each $b \in B$ where $a^\star$ refers the closest element of A to b, thus returning the lowest cost for matching b into A according to c. Therefore, we define $\sigma = \sum_{\forall b \in B} c(|b - a^\star|)$ and replace $|A \cap B|$ with $|B| - \sigma$ in Equation 8.4 to define our similarity function:

$$jSim(A,B,c) = \frac{|B| - \sigma}{|A| + \sigma} \tag{8.5}$$

The implementation of $jSim$ is presented in Algorithm 27 where A and B are two non-empty sets of values sorted in ascending order and c is one of our cost functions. Such assumptions come directly from our case study; in more general cases, Algorithm 27 can be modified to return zero if either A or B is empty as well as A and B can be sorted beforehand if they are not (rising, in this way, the overall computational complexity which is now linear).

First, the algorithm checks that $|B|$ is smaller than $|A|$ otherwise the two sets are swapped (line 3). This check is crucial since performing the matching from the smallest set to the largest one, and not vice-versa, limits the number of matching pairs between A and B so as to ensure that the numerator in Equation (8.5) does not get greater than the denominator and the returned similarity value lies in $[0, 1]$. As counter example, assume that $B = \{3, 6\}$ and $A = \{4\}$ while $c(a, b)$ simply returns 0 if $|a - b| < 5$ otherwise 1. Skipping line 3 in Algorithm 27 leads to having $jSim(A, B, c) = (2{-}0)/(1{+}0) = 2$ (since all pairwise absolute distances are less than the threshold 5 hence $\sigma \leftarrow 0$) while, enabling the swap-test, $jSim(A, B, c) = 0.5 < 1$ as required.

The algorithm finds the best match for each value in B by calculating the related cost against the closest element in A. To keep low the overall computational complexity, Algorithm 27 takes advantage of the sorted inputs

```
1  function jSim(A, B, c)
2      if |B| > |A| then
3          swap(A, B)
4      end
5      l ← −∞
6      r ← a_0
7      i ← 1
8      j ← 0
9      σ ← 0
10     while true do
11         while j < |B| ∧ b_j < r do
12             σ ← σ + c(min(b_j − l, r − b_j))
13             j ← j + 1
14         end
15         if i = |A| ∨ j = |B| then
16             break
17         end
18         l ← r
19         r ← a_i
20         i ← i + 1
21     end
22     while j < |B| do
23         σ ← σ + c(b_j − r)
24         j ← j + 1
25     end
26     return (|B| − σ)/(|A| + σ)
27 end
```

Algorithm 4: $jSim(A, B, c)$ computes the similarity between two non-empty sets of ordered values, $A = \{a_0, ..., a_{|A|-1}\}$ and $B = \{b_0, ..., b_{|B|-1}\}$, given a cost function c.

and logically divides A into a number of ranges defined by pairs of consecutive elements (identified by l, the left end-point, and r, the right end-point). As depicted in the example in Figure 8.2, the algorithm calculates the cost associated with the elements of B belonging to the current range $[l, r)$ by matching each of them with the closer end-point then adds the related cost to σ (line 12).

$$A = \{5, 7, 37, 237, 433, 630, 1685\}$$
$$B = \{5, 171, 1812\}$$

Figure 8.2. In this example, the red element is the only one in the blue range and it will be associated to the right end-point (i.e., $r = 237$) which is closer than the other one (i.e., $l = 37$).

As soon as the elements of B become greater than r, the range-endpoints are shifted forward to include the next A element (lines 18 and 19) and the matching process is repeated until the end of A or B is reached (line 16). In case A terminates before all elements of B have been matched, the cost for the remaining B elements is computed against the last element of A (line 23) which is straightforwardly the closest one. Once σ has been calculated, Algorithm 27 returns the similarity $jSim$ as defined by Equation 8.5 (line 26).

It turns out that each element of B is tested once while sliding the endpoints r and l over A while the cost for matching any remaining B elements is calculated directly sweeping the tail of B, hence the overall number of operations performed by Algorithm 27 is $O(|A|+|B|)$.

8.3.2 Group Identification

Followed by the similarity detection, let $jSimMtrx$ be the $n \times n$ matrix for $pts(m_{[i..n]})$ such that each row contains a pairwise similarity with all other sequences in the data set, in Algorithm 7 line 4. The matrix is then used to compute the distance between any of the two sequences such that $\Delta(pts(m_1), pts(m_2))$ using a linkage function. Since we opt to choose an unsupervised technique i.e., agglomerative hierarchical clustering, the linkage function is augmented with our customized $jSimMtrx$ and linkage type (which is $'complete'$ in our

case). We do not need to define the cut-off since the interest was real proximity of similarity. The function performs $n(n-1)/2$ comparisons to generate a 3-column *pairs* matrix, Algorithm 7 line 7. The *pairs* is then used to form the clusters with respect to the minimum distance between sequences and merge into a tree of n leaf nodes, the performance events. This multilevel hierarchy is then visualized with the dendrogram graphical tool which is well-known for qualitative and quantitative evaluations.

```
1 load n measurements into m_i ∈ M(p)
2 for i ← 1 to n do
3 |   for j ← 1 to n do
4 |   |   jSimMtrx_{i,j} < −jSim(pts(m_i), pts(m_j), c)
5 |   end
6 end
7 jSimClusters(jSimMtrx,'complete')
```

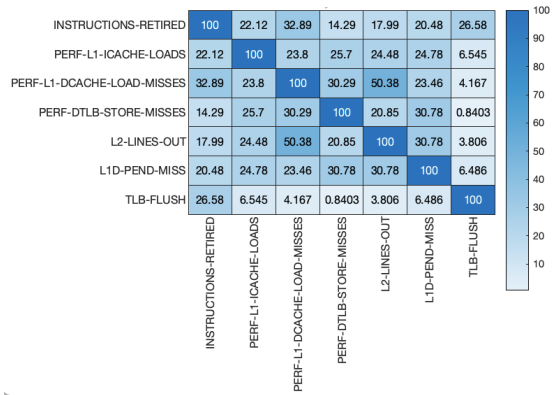
Algorithm 5: Identify clusters of similar performance events

8.4 Implementation and Experiments

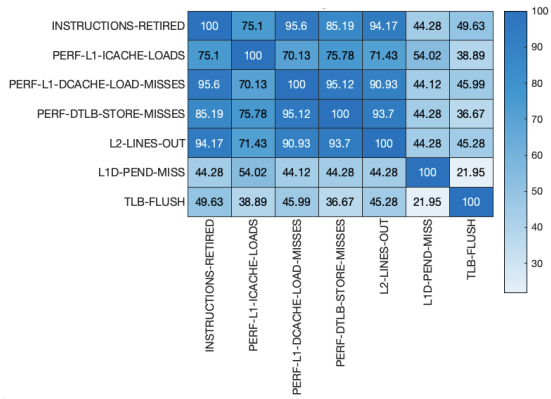
We measure the performance of the applications by using *PAPI library version 5.7.0.0* to sample the PMCs. Since the aim was to identify the groups of similar performance events so the entire execution period of the applications is measured for each event. To sample the applications a symmetric 5 milliseconds period was used which generates a varying number of samples depending on their execution time. From these time-based measurements, we detect segmentation points in each using *findchangepts()* function available in *Matlab version R2021*. Overall comparison, analysis, and visualization of grouped sequences are performed in Matlab.

Test Applications: We have chosen 6 different applications for the experiments based on their functions. We characterize 2×2 matrix multiplication, *meltdown* (a malware), *SUSAN* (image processor to find corners), *SIFT* (a complex feature detection algorithm to detects objects rather than just corners), *Multiresolution analysis kernel* (MADNESS) and *Covariance* (for Coefficient of

(a) Approximate similarity using C1



(b) Approximate similarity using C2



(c) Approximate similarity using C3

Figure 8.3. Pairwise proximity of similarity between some of the performance events of *Meltdown* application

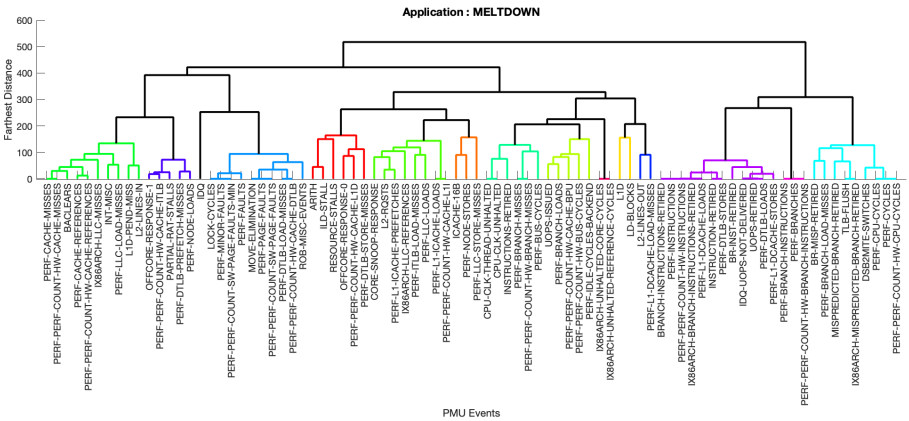
Variance Computation). The motive behind their selection is the significant use of computation functions and memory utilization in many industrial systems. Such applications can enormously impact the system performance due to their eager resource utilization demands.

Measurements: Each test application was characterized 20 times for its complete execution period using the solution provided in [6, 7]. For each application, a different number of performance events was captured since these events are coming from hardware, software and kernel. The hardware events may remain similar because they are coming from the same platform but the list of software events may vary since each application produces different events based on its distinct resource utilization demand.

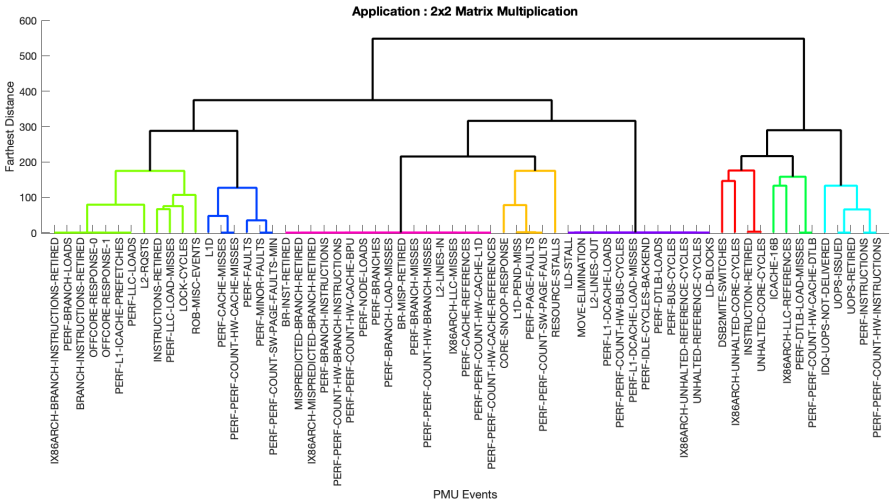
Results: We run the application and collect sequences of segmentation points as proposed in [7]. A few of them are listed in Table 8.1 with their corresponding performance event for application *meltdown*. Supplying a batch of sequences to our clustering algorithm, Algorithm 7, the similarity matrices for a subset of performance events are illustrated in Figure 8.3. A hierarchy is formed by merging the identified pairs using a similarity matrix and is presented in Figure 8.4.

Table 8.1. Sequences of segmentation points for some performance events of *Meltdown* application

Performance Event	**Sequence of Segmentation Points**
INSTRUCTIONS-RETIRED	{37, 48, 66, 70}
PERF-L1-ICACHE-LOADS	{3, 29, 52, 64, 70}
PERF-L1-DCACHE-LOAD-MISSES	{3, 37, 41, 69}
PERF-DTLB-STORE-MISSES	{3, 34, 36, 63}
L2-LINES-OUT	{3, 40, 58, 69}
L1D-PEND-MISS	{3, 5, 36, 39, 51, 58, 62, 67, 71}
TLB-FLUSH	{48, 73}



(a)



(b)

Figure 8.4. Clustering Results for Different Applications

8.5 Discussion

With respect to two major contributions of the paper, it was important to understand which cost function can establish a good base for the classification rule. The optimal number of classes will always be the unique data points if the classification rule is 'identical' only. Therefore, a prospective cost function is anticipated to compute a real-value distance from the closest match. Through visual observation, we evaluate c_2 yields a higher cost since it applies the penalty to the absolute distance between the points instead of a normalized cost up to the defined threshold. This is also apparent from Figure 8.3 and Table 8.1 that c_2 does not perform well when competing with c_1 and c_3 such as for performance events *INSTRUCTIONS-RETIRED* and *PERF-L1-ICACHE-LOADS*. The sequences of stated performance events are quite similar but c_2 identifies them as far distant. This also implies the fact that more rigorous costs are expected if the sequences are derived from the measurements consisting of thousands of data points. However, this means c_2 is relatively advantageous when strict or close matches are desired.

The above-stated observations stimulate further investigation between c_1 and c_3. The c_3 applies a sharp cost as soon as the distance between the points exceeds the threshold. This is good in a way to restrict the measure of similarity but in reality, there is always room for sampling bias and different loads of the execution environment within the captured performance data. In contrast, c_1 applies smoother cost not until the threshold is reached but also while leaving the boundaries of the similarity zone. For example, if the threshold of similarity is 25 and the distance between two points is 26 then it does not suddenly becomes dissimilar. Instead of a steep kick out from the similarity zone, a smooth departure is allowed. Such a smooth method is deemed appropriate when the data characteristics are complex and rational. Next, talking about *TLB-FLUSH* when compared with *L1D-PEND-MISS* it finds a close match for both elements yet the similarity is low, also shown in Figure 8.3a. The resultant similarity is nevertheless factual considering the difference in the cardinality of the compared sequences. Besides, the visual inspection of the results also verifies the level of identified similarity. Consequently, the performance of c_1 is recognized as accepted.

Moving on to the second contribution, Figure 8.4 illustrates the groups of similar performance events for applications *meltdown* and *2x2matrix multiplication*. The formulated hierarchy of different sequences demonstrate how distant they are, as shown in the case of *PERF-L1-DCACHE-LOAD-MISSES*, *PERF-DTLB-STORE-MISSES* and *L2-LINES-OUT* in Figure 8.4a. The dendrogram clearly presents that they might not be the same but comes in the same cluster with a certain distance.

Although the groups are not identified based on the magnitude of change, the visual analysis of clusters observes the change in behavior at a similar time for *BRANCH-INSTRUCTIONS-RETIRED*, *PERF-BRANCH-LOADS*, *PERF-L1-ICACHE-PREFETCHES* and *PERF-LLC-LOADS*, in Figure 8.4b. This information can be used to identify possible relevance between various resources such as processor and L1 cache. However, to detect impact and dependence more features are to be investigated such as the trend of data at the segmentation points.

8.6 Related Work

PMCs have also been used for behavioral-image formation where each performance event is considered as a feature [18]. The research includes features (PMCs) as images for behavioral analysis using a deep learning algorithm to know the normal or abnormal state of the system.

For finding similarity there are many existing approaches such as DNA similarity, cosine similarity, edit distance, and Jaccard index but they have preconditions such as identical or different lengths, same data structure or exact matches [12, 13, 14]. The way they compare is more strict and can be applied in absolute conditions. When it is not the case researchers like Fletcher and Islam [19] have used the Jaccard index for comparing patterns coming from different techniques. Their proposed method converts each pattern into a single element which is also the commonality between their and our solution. However, our method to get a discrete value of similarity is different. Their method translates each pattern into an element of its own set whereas we compute the similarity based on element-wise weighted distance with respect to the lengths of the sequences. This is an additional strength of our proposed mechanism to

handle the inconsistencies of data.

A similar approach has also been applied by Koch, Zemel and Salakhutdinov [20] for one-shot image recognition where very limited or sometimes single example is available to compare in supervised machine learning. They employed the sigmoid function in convolutional neural networks to find the similarity between the final and hidden layers of the twin network. The approach was to scale the absolute distance between 0 and 1 with the help of training parameters. Since their problem was binary classification so instead of utilizing real-value output the values from 0.5 to 1 were taken as dissimilar. Whereas we use the resultant weighted cost as a probability of similarity. Moreover, their working sets were of the same length so one-to-one comparisons were directly possible which on the contrary was not a viable option for us. So we provide additional functionality to find the closest possible match with our holistic and intelligent approach.

8.7 Conclusion and Future Work

We have presented a mechanism that can compute the proximity of similarity between ordered sequences of uneven lengths. The method based on weighted real-value costs nicely handles the measure of dissimilarity. The method is also flexible to choose between firm, rigorous and strict penalties based on the needs of how strict or moderate comparisons are to be performed. We outline the method that applies the appropriate cost by investigating the closest match. The mechanism was able to group different performance events based on the segmentation points. We have also argued the possible leads towards identifying relations and dependence between different performance events.

Lastly, we continue toward automatically creating an application fingerprint based on its resource utilization for detection, identification or even decision-making. An immediate extension can be relating the trends in the data before and after the segmentation points to identify the impact between different resources.

Bibliography

[1] Intel. Intel® 64 and ia-32 architectures software developer's manual. Technical report, Intel, 2022.

[2] Ella Bingham, Aristides Gionis, Niina Haiminen, Heli Hiisilä, and Heikki Mannila. Segmentation and dimensionality reduction. In *2006 SIAM International Conference on Data Mining*, pages 372–383. Society for Industrial and Applied Mathematics, 2006.

[3] MathWorks. findchangepts - Find abrupt changes in signal, 2023.

[4] SAMIR BEN HARIZ, JONATHAN J. WYLIE, and QIANG ZHANG. Optimal rate of convergence for nonparametric change-point estimators for nonstationary sequences. 2007.

[5] Dengsheng Zhang. *Fundamentals of image data mining*. Springer International Publishing, 2nd edition, 2019.

[6] Shamoona Imtiaz, Jakob Danielsson, Moris Behnam, Gabriele Capannini, Jan Carlson, and Marcus Jägemar. Automatic platform-independent monitoring and ranking of hardware resource utilization. In *26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2021.

[7] Shamoona Imtiaz, Moris Behnam, Gabriele Capannini, Jan Carlson, and Marcus Jägemar. Automatic segmentation of resource utilization data. In *1st IEEE Industrial Electronics Society Annual On-Line Conference (ONCON 2022*, pages 1–6. IEEE, 2022.

[8] Brendan Gregg. *Systems performance: enterprise and the cloud*. Pearson, 2nd edition, 2020.

[9] AMD. Open-source register reference for amd family 17h processors models 00h-2fh, 2018.

[10] ARM. Arm architecture reference manual - armv8, for armv8-a architecture profile.pdf, 2017.

[11] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A portable interface to hardware performance counters. In *Proceedings of the department of defense HPCMP users group conference*, volume 710, 1999.

[12] Matthias Studer and Gilbert Ritschard. What matters in differences between life trajectories: a comparative review of sequence dissimilarity measures. *Journal of the Royal Statistical Society: Series A (Statistics in Society), 179: 481-511,*, 179(2):481–511, 2016.

[13] Alfirna Rizqi Lahitani, Adhistya Erna Permanasari, and Noor Akhmad Setiawan. Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management*, pages 1–6. IEEE, 2016.

[14] Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. Using of jaccard coefficient for keywords similarity. In *International multiconference of engineers and computer scientists*, volume 1, pages 380–384. IEEE, 2013.

[15] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task. In *rXiv preprint arXiv:1804.02763*, 2018.

[16] Matlab. Choose cluster analysis method, 2023.

[17] Allan H. Murphy. The finley affair: A signal event in the history of forecast verification. *Weather and Forecasting*, 11(1):3 – 20, 1996.

[18] Gaddisa Olani Ganfure, Chun-Feng Wu, Yuan-Hao Chang, and Wei-Kuan Shih. Deepware: Imaging performance counters with deep learning to detect ransomware. In *IEEE Transactions on Computers*, volume 72, pages 600–613, 2022.

[19] Sam Fletcher and Md Zahidul Islam. Comparing sets of patterns with the jaccard index. *Australasian Journal of Information Systems*, 22, 2018.

[20] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. *ICML deep learning workshop*, 2(1), 2015.