In 28th Annual Conference of the IEEE Industrial Electronics Society (ETFA2023), September 12–15, 2023, Sinaia, Romania.

Automatic Clustering of Performance Events

Shamoona Imtiaz[1], Gabriele Capannini[1], Jan Carlson[1], Moris Behnam[1], Marcus Jägemar[2]
[1] Mälardalen University, Västerås, Sweden, [2] Ericsson AB, Stockholm, Sweden
[1]first.last@mdu.se, [2]first.last@ericsson.com

***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.**

***Index Terms*—Performance monitoring counters, Performance events, Hierarchical clustering, Sigmoid function, Similarity measurement, Similarity detection, Change point detection.**

I. 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.

Acknowledgements: This work was supported by the Knowledge Foundation in Sweden through the ACICS project (20190038).

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:

a) **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.

b) **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 II for the readers to easily understand the contribution made through this work. Next, we present our proposed solution in Section III describing the approach used to achieve the goal of the study. The implementation details and the experimental setup is then outlined in Section IV. Following the implementation details, results are discussed in Section V to extend the reader's knowledge. The state-of-the-art and related work to our study is then presented in Section VI. Finally, the anticipated future work followed by the conclusion concludes the paper in Section VII.

II. 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.

A. 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.

B. 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.

C. 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:

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

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.

D. 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.

E. 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.

III. Proposed Solution

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

A. 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.

a) **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.

b) **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.

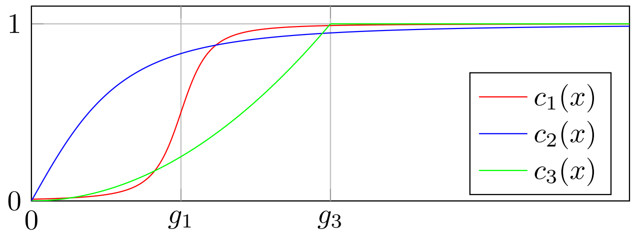


Fig. 1. Plot of the cost functions proposed with the thresholds g_1 and g_3.

*c) **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 1 where $x = |p_1 - p_2|$ and defined as:

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

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

$$c_3(x) = \min(1, (g_3 x)^2) \tag{3}$$

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 1, 2 and 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:

- The first cost function c_1, Equation 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 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 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 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.

*d) **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 similarity [17] defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4}$$

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 4 to define our similarity function:

$$jSim(A, B, c) = \frac{|B| - \sigma}{|A| + \sigma} \tag{5}$$

The implementation of $jSim$ is presented in Algorithm 1 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 1 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 (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 1 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 1 takes advantage of the sorted inputs 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 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 11).

As soon as the elements of B become greater than r, the range-endpoints are shifted forward to include the next

Algorithm 1: $jSim(A, B, c)$ 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.

```
1  function $jSim(A, B, c)$
2      if $|B| > |A|$ then
3          swap$(A, B)$
4      $l \leftarrow -\infty$
5      $r \leftarrow a_0$
6      $i \leftarrow 1$
7      $j \leftarrow 0$
8      $\sigma \leftarrow 0$
9      while true do
10         while $j < |B| \wedge b_j < r$ do
11             $\sigma \leftarrow \sigma + c(\min(b_j - l, r - b_j))$
12             $j \leftarrow j + 1$
13         if $i = |A| \vee j = |B|$ then
14             break
15         $l \leftarrow r$
16         $r \leftarrow a_i$
17         $i \leftarrow i + 1$
18     while $j < |B|$ do
19         $\sigma \leftarrow \sigma + c(b_j - r)$
20         $j \leftarrow j + 1$
21     return $(|B| - \sigma)/(|A| + \sigma)$
```

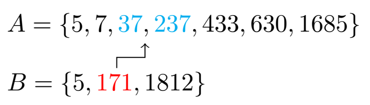


Fig. 2. In this example, the red element is the only one in the blue range and it will be associated to the right end-point (i.e., $r = 237$) which is closer than the other one (i.e., $l = 37$).

A element (lines 15 and 16) and the matching process is repeated until the end of A or B is reached (line 14). 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 19) which is straightforwardly the closest one. Once σ has been calculated, Algorithm 1 returns the similarity *jSim* as defined by Equation 5 (line 21).

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 1 is $O(|A|+|B|)$.

B. 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 2 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 2 line 5. 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.

Algorithm 2: Identify clusters of similar performance events

```
1  load $n$ measurements into $m_i \in M(p)$
2  for $i \leftarrow 1$ to $n$ do
3      for $j \leftarrow 1$ to $n$ do
4          $jSimMtrx_{i,j} < -jSim(pts(m_i), pts(m_j), c)$
5  $jSimClusters(jSimMtrx, 'complete')$
```

IV. 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 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 I with their corresponding performance event for application *meltdown*. Supplying a batch of sequences to

(a) Approximate similarity using C1 (b) Approximate similarity using C2 (c) Approximate similarity using C3

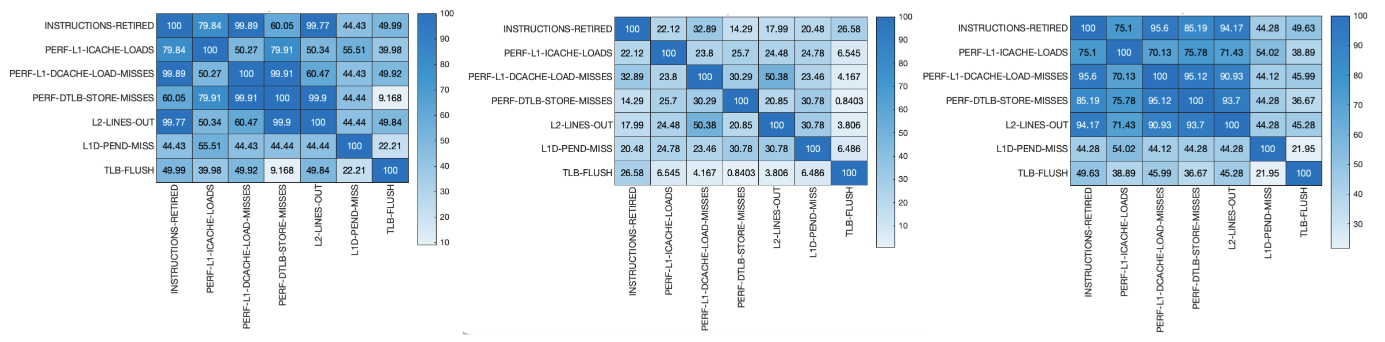


Fig. 3. Pairwise proximity of similarity between some of the performance events of *Meltdown* application

our clustering algorithm, Algorithm 2, the similarity matrices for a subset of performance events are illustrated in Figure 3. A hierarchy is formed by merging the identified pairs using a similarity matrix and is presented in Figure 4.

TABLE I
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}

V. Discussion

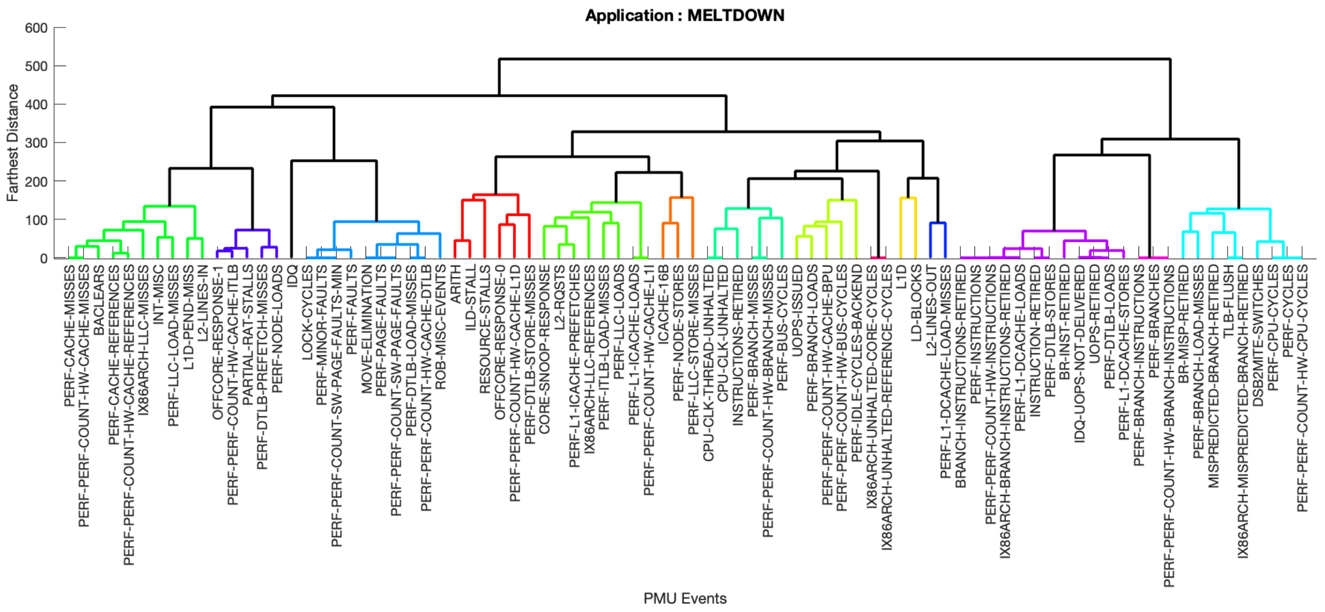
With respect to two major contributions of the paper, it was important to understand which cost function can establish a good base for the classification rule. The optimal number of classes will always be the unique data points if the classification rule is 'identical' only. Therefore, a prospective 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 3 and Table I 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 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 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 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 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.

(a)



(b)

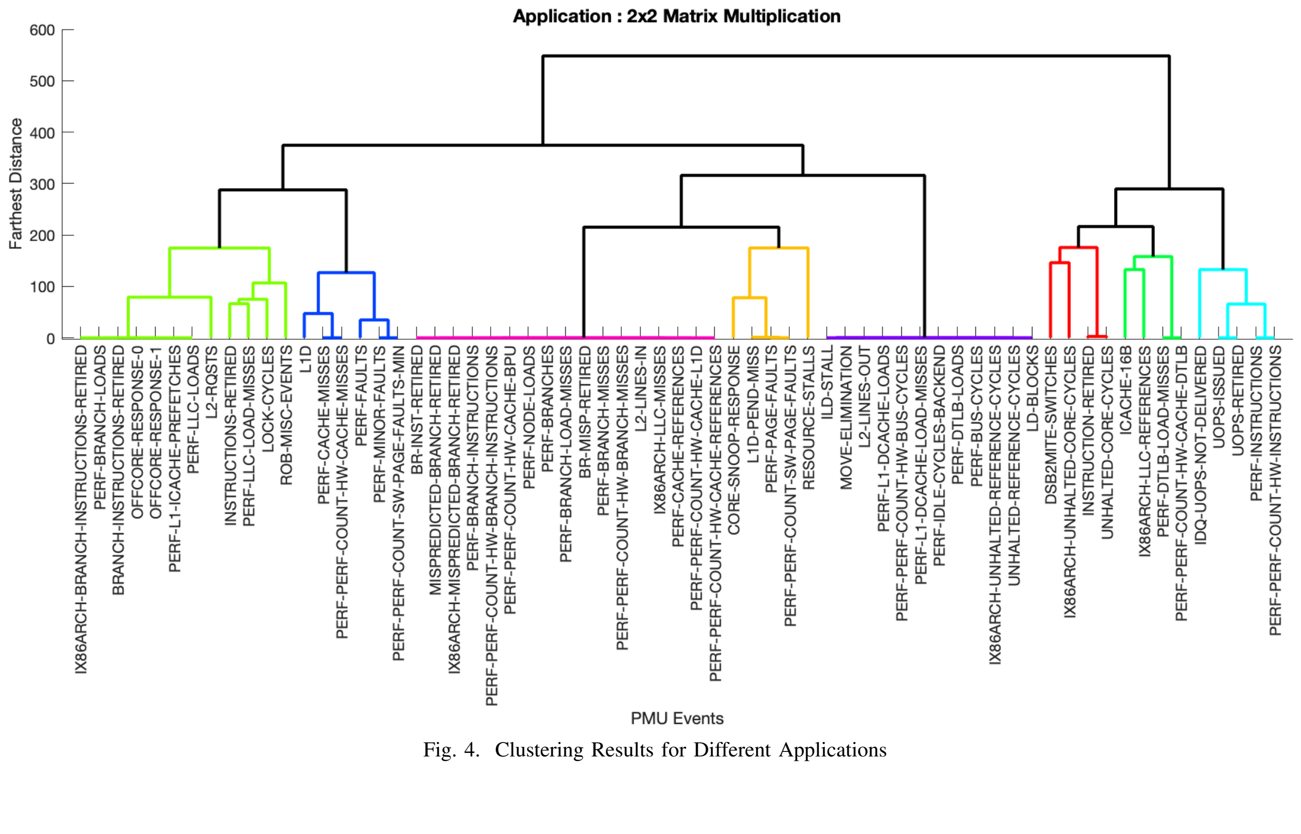


Fig. 4. Clustering Results for Different Applications

VI. 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.

VII. Conclusion and Future Work

We have presented a mechanism that can compute the proximity of similarity between ordered sequences of uneven lengths. The met hod 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.

References

[1] Intel, "Intel® 64 and ia-32 architectures software developer's manual," Intel, Tech. Rep., 2022.

[2] E. Bingham, A. Gionis, N. Haiminen, H. Hiisilä, and H. Mannila, "Segmentation and dimensionality reduction," in *2006 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 2006, pp. 372–383.

[3] MathWorks. (2023) findchangepts - Find abrupt changes in signal. [Online]. Available: https://ch.mathworks.com/help/signal/ref/findchangepts.html

[4] S. B. Hariz, J. J. Wylie, and Q. Zhang, "Optimal rate of convergence for nonparametric change-point estimators for nonstationary sequences." 2007, pp. 1802–1826.

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

[6] S. Imtiaz, J. Danielsson, M. Behnam, G. Capannini, J. Carlson, and M. Jägemar, "Automatic platform-independent monitoring and ranking of hardware resource utilization," in *26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021, pp. 1–8.

[7] S. Imtiaz, M. Behnam, G. Capannini, J. Carlson, and M. Jägemar, "Automatic segmentation of resource utilization data," in *1st IEEE Industrial Electronics Society Annual On-Line Conference (ONCON 2022)*. IEEE, 2022, pp. 1–6.

[8] B. Gregg, *Systems Performance : Enterprise and the Cloud Second Edition*, 2nd ed. Pearson, 2020.

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

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

[11] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "PAPI: A portable interface to hardware performance counters," in *Proceedings of the department of defense HPCMP users group conference*, vol. 710, 1999.

[12] M. Studer and G. 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,*, vol. 179, no. 2, pp. 481–511, 2016.

[13] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, "Cosine similarity to determine similarity measure: Study case in online essay assessment," in *2016 4th International Conference on Cyber and IT Service Management*. IEEE, 2016, pp. 1–6.

[14] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in *International ulticonference of engineers and computer scientists*, vol. 1, no. 6. IEEE, 2013, pp. 380–384.

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

[16] Matlab. (2023) Choose cluster analysis method. [Online]. Available: https://se.mathworks.com/help/stats/choose-cluster-analysis-method.html

[17] A. H. Murphy, "The finley affair: A signal event in the history of forecast verification," *Weather and Forecasting*, vol. 11, no. 1, pp. 3 – 20, 1996.

[18] G. O. Ganfure, C.-F. Wu, Y.-H. Chang, and W.-K. Shih, "Deepware: Imaging performance counters with deep learning to detect ransomware," in *IEEE Transactions on Computers*, vol. 72, no. 3, 2022, pp. 600–613.

[19] S. Fletcher and M. Z. Islam, "Comparing sets of patterns with the Jaccard index," *Australasian Journal of Information Systems*, vol. 22, 2018.

[20] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," *ICML deep learning workshop*, vol. 2, no. 1, 2015.