In 1st IEEE Industrial Electronics Society Annual On-Line Conference (ONCON) 2022, December 09–11, 2022.

Automatic Segmentation of Resource Utilization Data

Shamoona Imtiaz[1], Moris Behnam[1], Gabriele Capannini[1], Jan Carlson[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*—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.**

I. 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 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

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

979-8-3503-9806-9/22$31.00

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 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 II to provide relevant knowledge required to easily understand the contribution made through this work. Next, Section III explains the proposed approach to achieve the goal of the study. Section IV extends the readers knowledge for implementation details and experimental setup. After successful implementation, results are being discussed in Section V. We also relate our work to the state-of-the-art in Section VI. Finally the anticipated future work followed by the conclusion wraps up the paper in Section VII.

II. 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 II-A, Section II-B, Section II-C respectively.

A. 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.

B. 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.

C. 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.

III. 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.

A. 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 1. 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 Segments:** 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 III.

B. 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 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.

IV. 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

Algorithm 1: Find segments for PMU event e

```
r contains x measurements in R_e
for j = 1 to x do
    for t = 1 to maxThresh do
        ⟨resError, pts⟩ = findchangepts(r[j], t)
        /* tr contains primary threshold
           for one measurement of PMU
           event e                          */
        if t == 1 then
            tr[j] = t
            re[j] = resError
        if resError == lastResError then
            tr[j] = t
            re[j] = resError
            break
        lastResError = resError
/* th contains optimal threshold for
   all measurements of PMU event e  */
th = computeOptimalThresh(tr, re)
/* Find segments using optimal
   threshold                        */
for j = 1 to x do
    ⟨resError, pts⟩ = findchangepts(r[j],th)
    S[j].pts = pts
    S[j].resError = resError
/* Find optimal threshold           */
function computeOptimalThresh(t, re)
    med = median of re
    thresh = 0
    for j = 1 to length(re) do
        if (re[j] == med) or
         (re[j] < med and re[j+1] > med) then
            thresh = t[j]
    return thresh
```

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

TABLE I
ACCURACY OF SEGMENT DETECTION METHOD

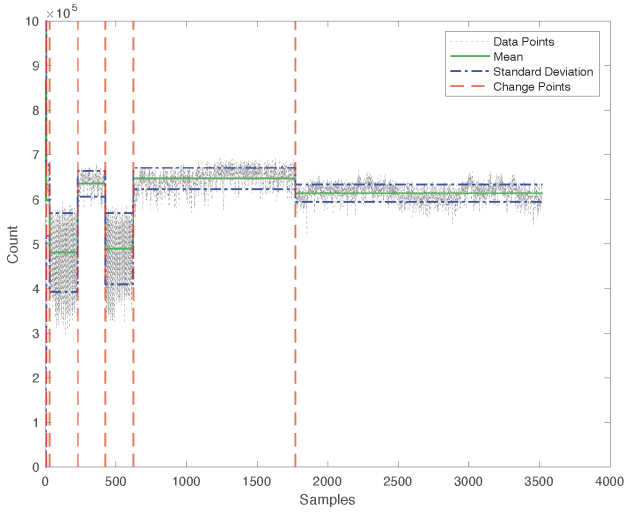
PMU Event	Coefficient of Variance (%)	Maximum Residual Error
Branch Instructions Retired	2.63	1.0130e+05
Instructions Retired	2.66	1.1767e+05
L1D_PEND_MISS	1.79	1.3675e+05
L1D	1.68	1.0144e+05
TLB_FLUSH_DATA	1.98	1.1829e+05
perf-CPU-CYCLES	4.11	1.2375e+05

V. DISCUSSION

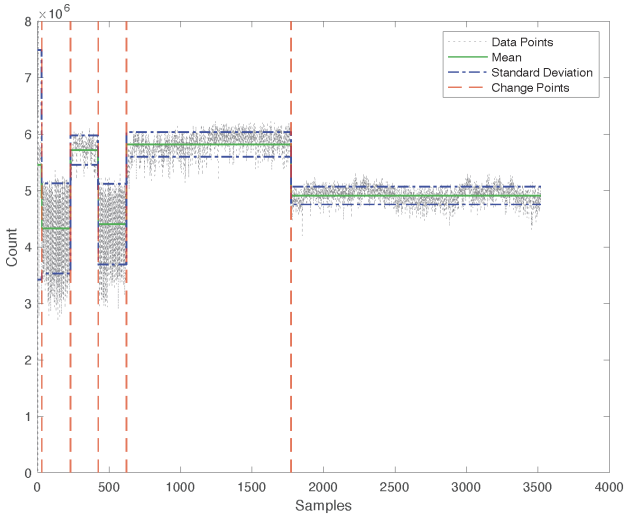
The proposed dynamic approach automatically detects number and location of segments based on root means square level. The method does not need to know the number of change points as an input parameter to find segments. This sequential method takes the complete data series into account to be able to iteratively investigate and adjust key points until the residual error becomes minimum. The results in Figure 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 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 I 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 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.

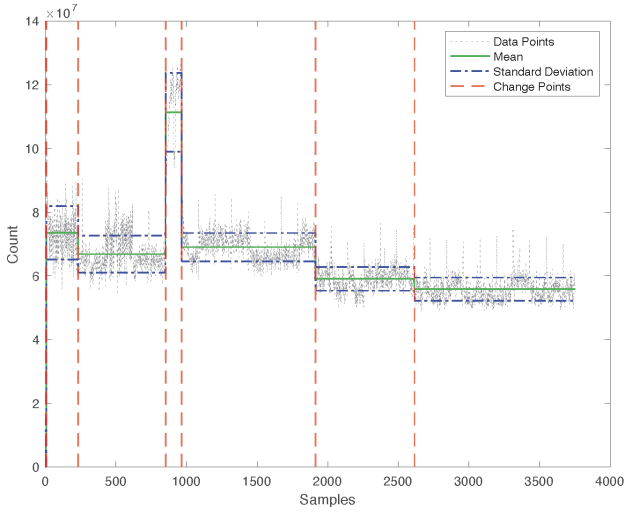
In Figure 1(d), last segment of *L1D* shows consistent higher L1D cache utilization starting roughly from sample 600 to 3700. This knowledge can be useful in the case of hyper-threading which allows to run more than one thread on each core. Applications running on a hyperthreaded CPU utilize two hardware 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,



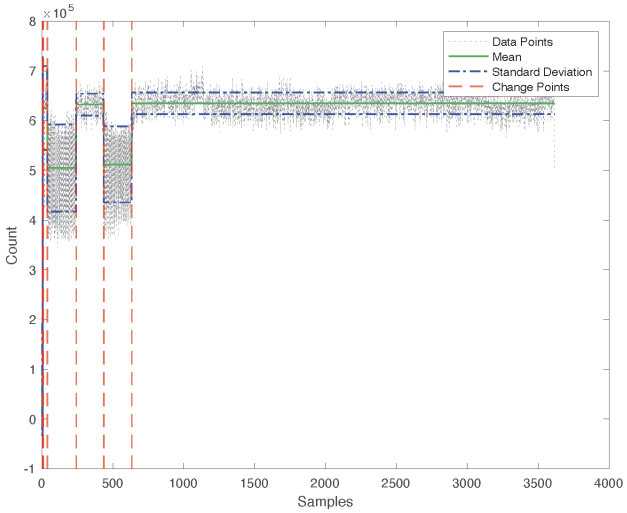
(a) Segments for Branch Instructions Retired



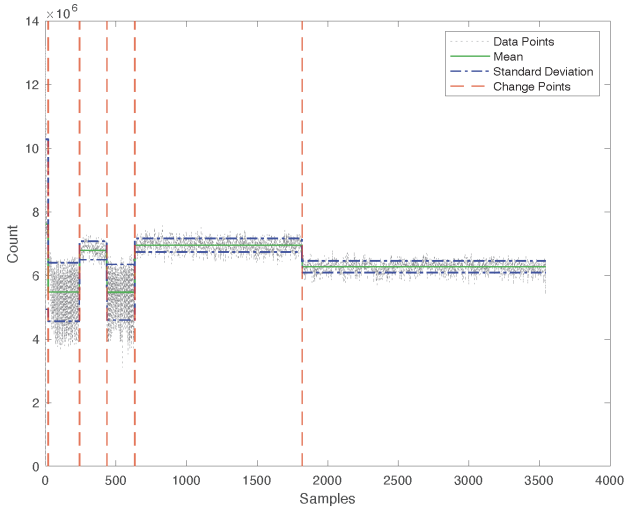
(b) Segments for Instructions Retired



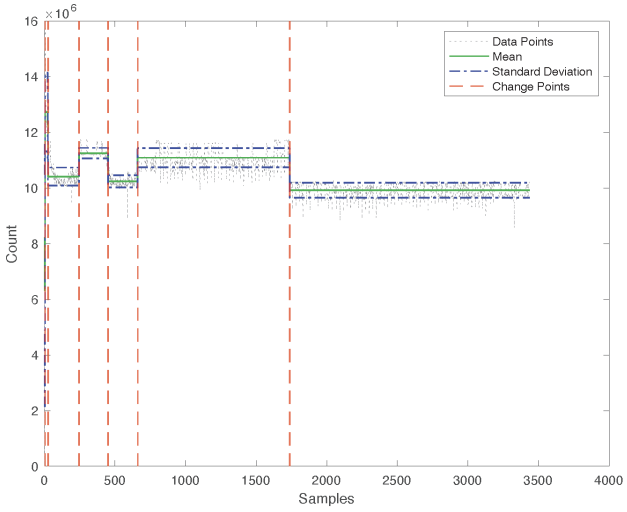
(c) Segments for L1D_PEND_MISS



(d) Segments for L1D



(e) Segments for TLB_FLUSH



(f) Segments for perf-CPU-CYCLES

Fig. 1. Simulation results for the network.

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 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.

VI. 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.

VII. 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.

References

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

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

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

[4] M. Lavielle, "Using penalized contrasts for the change-point problem," *Signal processing*, vol. 85, no. 8, pp. 1501–1510, 2005.

[5] A. Pro, "How change point detection works," 2022. [Online]. Available: https://pro.arcgis.com/en/pro-app/2.9/tool-reference/ space-time-pattern-mining/how-change-point-detection-works.htm

[6] S. Sharma, D. A. Swayne, and C. Obimbo, "Trend analysis and change point techniques: a survey," in *Energy, Ecology and Environment*, vol. 1, no. 3, 2016, pp. 123–130.

[7] A. Sembrant, D. Eklov, and E. Hagersten, "Efficient software-based online phase classification," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2011, pp. 104–115.

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

[9] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, and perils of using hardware performance counters for security," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 20–38.

[10] Y. Wang, C. Wu1, Z. Ji, B. Wang, and Y. Liang, "Non-parametric change-point method for differential gene expression detection," *PloS one*, vol. 6, no. 5, pp. e20 060–e20 060, 2011.

[11] MathWorks. (2022) findchangepts - find abrupt changes in signal. [Online]. Available: https://se.mathworks.com/help/signal/ref/findchangepts.html #mw_748d5529-e05a-4b55-a3fe-2a12a5772d22

[12] Indeed. (2022) What is variance? definition and how to calculate it. [Online]. Available: https://www.indeed.com/career-advice/career-development/what-is-variance

[13] S. Imtiaz, J. Danielsson, M. Behnam, G. Capannini, J. Carlson, and M. Jäagem, "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.

[14] D. S. Matteson and N. A. James, "A nonparametric approach for multiple change point analysis of multivariate data," *Journal of the American Statistical Association, 109:505, 334-345,*, vol. 109, no. 505, p. 34–345, 2012.

[15] Y.-C. Yao, "Estimating the number of change-points via schwarz'criterion," in *Statistics & Probability Letters*, vol. 6, no. 3, 1988, pp. 181–189.

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