A Machine Learning-Based Framework for Throughput Estimation of Time-Varying Applications in Multi-Core Servers

Arman Iranfar*, Wellington Silva De Souza†, Marina Zapater*, Katzalin Olcoz‡, Samuel Xavier de Souza†, David Atienza*
*Embedded Systems Laboratory (ESL), Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland
†Universidade Federal do Rio Grande do Norte, Brazil
‡Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Spain
*{arman.iranfar, marina.zapater, david.atienza}@epfl.ch, †{wellingtonsouza,samuel}@{imd,dca}.ufrn.br, ‡katzalin@ucm.es

***Abstract*—Accurate workload prediction and throughput estimation are keys in efficient proactive power and performance management of multi-core platforms. Although hardware performance counters available on modern platforms contain important information about the application behavior, employing them efficiently is not straightforward when dealing with time-varying applications even if they have iterative structures. In this work, we propose a machine learning-based framework for workload prediction and throughput estimation using hardware events. Our framework enables throughput estimation over various available system configurations, namely, number of parallel threads and operating frequency. In particular, we first employ workload clustering and classification techniques along with Markov chains to predict the next workload for each available system configuration. Then, the predicted workload is used to estimate the next expected throughput through a machine learning-based regression model. The comparison with state of the art demonstrates that our framework is able to improve Quality of Service (QoS) by 3.4x, while consuming 15% less power thanks to the more accurate throughput estimation.**

I. Introduction

The advent of next-generation Quality-of-Service (QoS)-sensitive applications require novel approaches to deal with power, performance, and time-predictability challenges [1]. To overcome these challenges, deploying the hardware and software efficiently is vital.

Multi-core platforms play a major role in achieving the required performance through parallel processing. However, these platforms include more complex software and hardware, which pose extra challenges for power management and time-predictability of applications. Therefore, finding the optimal number of processors that satisfies the target QoS, while minimizing power consumption is challenging. In addition, the workload variations within an application execution time exacerbates the aforementioned challenge. Moreover, while Dynamic Voltage and Frequency Scaling (DVFS) makes dynamic power management and performance control feasible, numerous frequencies available in modern processors considerably enlarge the design space. In addition, DVFS is agnostic to the number of threads assigned to the application. Thus, due to these workload variations within an application, the selection of the optimal operating frequency for such a large design space is very complex.

In order to tackle such a large design space, modern processors are equipped with hardware performance counters that enable non-intrusive performance monitoring of the processors [2]. Indeed, these counters can provide important information about workload behavior, which can be used in workload prediction. This workload prediction, if accomplished accurately enough, can be effectively deployed for power management and performance control of time-varying QoS-sensitive applications. To this end, hardware event-based simulators and analytic models have been traditionally used for workload assessment and prediction. However, due to the need for fast and dynamic changes of system configurations for efficient proactive power and performance management, these approaches are not feasible in practice [3].

This work has been supported by Spanish MINECO (GA. No. TIN2015-65277-R), the Spanish MINECO (GA. No. S2018/TCS-4423), the SERI Seed Money project (GA No. SMG1702), the EC H2020 RECIPE project (GA No. 801137), and the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657).

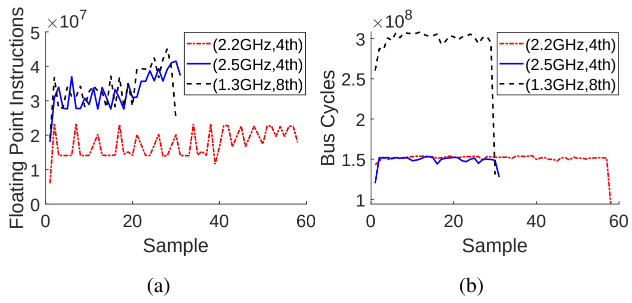


Fig. 1. Number of (a) floating point instructions and, (b) bus cycles, every 400 ms, under 3 system configurations: (frequency (GHz), number of threads)

Moreover, in multi-core platforms, information extracted from the performance counters does not purely indicate the workload behavior, as the collected data are also affected by the specific system configuration (i.e., the number of processors and the operating frequency). Therefore, workload prediction is not straightforward since with any change in the system configuration, performance counters may imply a new workload for the same input of the application. As an example, Fig. 1 shows the trace of two performance counters while running a time-varying application (from the beginning to the end) under three different configurations on a multicore server. For this specific application, the number of *Bus Cycles* are mainly affected by the number of threads rather than the core frequency. Furthermore, as expected, system configurations of $(2.5, 4)$ and $(1.3, 8)$ can provide almost the same average performance, with the former having twice *Bus Cycles*. Therefore, in this case, both measures are required to properly adapt the system configuration.

Nonetheless, when changing the system configuration to the other, the already-gathered hardware events cannot be directly used to predict the next values through conventional workload prediction techniques such as Auto-Regressive Integrated Moving Average (ARIMA) model and its derivatives [4]. On the contrary, machine learning (ML)-based approaches represent a promising solution, as they can directly learn from the data rather than counting on rule-based programming. More importantly, ML-based approaches are known to provide scalable solutions for large design spaces with low-overhead inference time, which make them appropriate candidates for power and performance management of multi-core platforms.

In this work, we propose an ML-based framework for workload prediction and throughput estimation of time-varying applications in multi-core platforms. In particular, the main contributions of our work are as follows:

- we propose a low-overhead workload prediction approach based on the hardware events. This approach can be used with any state-of-the-art many-core platform today.
- we develop a framework which provides the future throughput estimate under different system configurations using machine learning,
- we show how accurate workload prediction and throughput estimation per system configuration can save power in a multi-core server. Indeed, on average, our proposed framework provides 15% less power consumption with 3.4x less QoS violations compared to the state of the art.

II. Related Work

Workload prediction methodologies are quite rich in literature, especially for cloud-based applications. However, since these applications usually show workload variations on an hourly, daily, or even weekly basis, many of the previous works use ARIMA model and its derivatives [5], which is not necessarily compatible with highly time-varying non-cloud applications. For the latter, clustering, classification, and regression models have been widely used to predict the future workload for different purposes [6], [7]. Despite such a variety of applications, workload prediction in many-core platforms is mainly used for dynamic power and performance management.

The correlation of hardware events with performance and power consumption of the application has been traditionally used in literature for performance and power analysis, prediction, and tuning [8]. Although performance counters have been used for different application domains, the most common one is power and performance prediction and management [9]. For instance, Borghesi et al. [10] leverage hardware events as a part of the input to their predictive model for power consumption. These works, however, do not address power and performance estimation for different system configurations.

There are a few works that leverage hardware events for power and performance estimation across different system configuration. In particular, [11] develops a power estimation and thread scheduling scheme using a piece-wise model based on multiple linear regression. Nevertheless, this work is limited to thread scheduling as their only design parameter. Moreover, the scalability of the proposed strategy is in question due to the use of linear regression [3]. Besides, authors in [12] propose an online strategy for power and performance management of multithreaded applications using hardware event-based prediction. In their work, however, operating frequency as one of the main design space scaling factors has been ignored. More importantly, the scheme of [12] is based on linear regression which is known to be limited in modeling large design spaces and runtime hardware changes. Finally, in contrast to [11], [12], Wu et. al [3] provide low-overhead scalable solution by training a neural network with performance counter values to predict power consumption and performance of different applications at different GPU configurations. Authors, however, train their neural network with the overall behavior of complete input kernels rather than with fine-grained input samples. Such an approach can lead to wrong decisions for dynamic configuration selection, as it ignores rapid workload variations within a kernel, which is very typical in latest time-varying applications with iterative structures (cf. Fig. 3), such as, latest standards for video streaming.

In contrast to [3], our framework is able to provide fine-grained workload prediction across different system configurations to provide more accurate throughput estimation which can ultimately result in less power consumption.

III. Proposed Framework

Fig. 2 shows an overview of our proposed machine learning-based framework. The goal of our work is to estimate the future throughput of an application with iterative structure (as shown in Fig. 3) for different system configurations (i.e., core frequency and number of cores), given a set of hardware performance counters ($PC_j(t)$) at time t, obtained from a particular system configuration (j). Since at time t, the only available information is $PC_j(t)$, it is not straightforward to estimate the future throughput for all other configurations directly. Therefore, first, $PC_j(t)$ is used to specify the current workload ($WL_j(t)$) of the application through clustering and classification algorithms. This workload accounts for input and system configuration. Then, $WL_j(t)$, in conjunction with the previous workload of the application under the same configuration ($WL_j(t-1)$), is used to predict the next workload for all available system configurations, $Config_k(t+1)$. Next, the predicted workload ($WL_k^{'}(t+1)$) is employed to estimate the future throughput of the application for each configuration.

When using performance counters, the first step is to determine the most relevant counters that carry useful information about system and application. Therefore, in this section, we first explain how to choose these performance counters. Second, we explain in detail the steps we take for the per-configuration throughput estimation depicted in Fig. 2.

A. Counters Selection

Although there can be hundreds of counters available on a machine [13], the number of counters that can be simultaneously collected without time multiplexing are limited. Also, precision of the measured counters degrades when more counters are to be read. Moreover, using more counters as the system and application features for estimating the future throughput does not necessarily improve the accuracy. Finally, using redundant features may ultimately lead to more runtime overhead due to the increased complexity. Therefore, it is vital to select counters that carry the most relevant information regarding the target system and application behavior.

For each application in control, starting from the main hardware counters introduced in the literature [12], we use

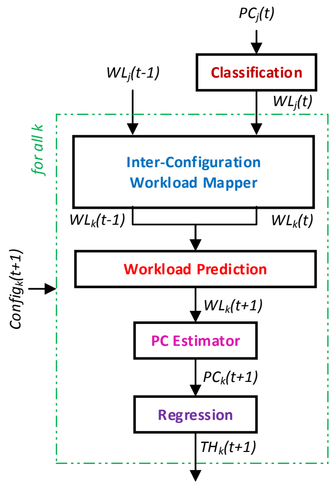


Fig. 2. Proposed ML-based framework

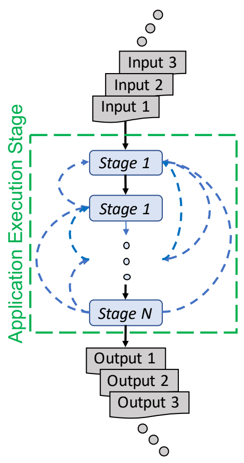


Fig. 3. Example of an application with iterative structure

the Pearson Correlation [14] to determine those that contain the most relevant information according to the throughput. For our purpose, these counters are the ones with larger correlation coefficient with respect to the throughput, which are less correlated with each other to avoid redundant information.

B. Workload Clustering and Classification

For a single application, the values of the performance counters and the application's throughput vary within different system configurations. Moreover, for the same system configuration, these values can change due to the workload variations at different stages of the application. Since with the same application and input size, each system configuration may lead to a different set of performance counters and throughput for a particular stage of the whole application, the workload type under process should be studied per configuration. Therefore, we propose to qualify WL_k based on PC_k for each $Config_k$.

With a constant performance counter sampling rate, a different number of workload types can be observed for each configuration. For configurations with higher frequency and larger number of cores, the sampling rate can be comparable to the execution time of one iteration of the application, while with lower frequency and smaller number of cores, it can be much higher than the execution time. Hence, for the latter, $PC(t)$ may indicate the difference between different stages of the application while for the former, $PC(t)$ may only show the difference between different inputs. Imagine a video encoding application that outputs one frame per second under a particular configuration. In such a case, with a sampling rate of 10 Hz, 10 sets of performance counters exist for a single frame. On the contrary, if with a different configuration the application outputs 10 frames per second, there would be one set of performance counter samples per frame.

In order to qualify the workload type per configuration, we propose to use the $Kmeans++$ clustering algorithm [15]. We consider the Silhouette [16] criterion to find the optimal number of clusters per configuration. Thereafter, considering the Random Forest (RF) [17] with 50 weak learners as the classification algorithm, we train an ensemble of bagged classification trees using the performance counter samples as the input features and the workload type (WL) obtained from the clustering as the output labels. While these two phases are performed offline, the obtained classification model is employed at runtime to qualify $WL_j(t)$ from $PC_j(t)$. We note that, RF suits well our purpose since we need to deal with a multi-class classification problem rather than a binary one. Moreover, RF is suitable when coping with a mixture of numerical and categorical features which leads us to use the features (performance counters) on their own various scales.

C. Inter-Configuration Workload Matching and Prediction

Once $WL_j(t)$ is recognized from $PC_j(t)$, for each available configuration ($Config_k(t+1)$) the next workload should be predicted. We propose to create a discrete-time, finite-state Markov chain [18] from workload transitions for each configuration using the same training data as in the workload classification. Thus, the next workload can be predicted considering the probability of transitioning to a new workload according to an already-observed chain of transitions. During runtime, we keep updating the transition probabilities to enhance the next workload prediction within a single configuration. We note that the length of this chain depends on (i) the application nature, (ii) the frequency of the workload changes within a certain configuration, and (iii) the performance counter sampling frequency. For our purpose, we found 10 Hz sampling rate and a chain length of two sufficient to account for workload variations without adding extra load to the system.

The $Kmeans++$ algorithm finds the optimal centroid seeds by first choosing seeds from a random observation of the data set. Consequently, as we run the algorithm for each configuration, the outcome clusters of one configuration are not in the same order as the clusters obtained for another configuration. For instance, the first cluster of the j^{th} configuration may correspond to any workload type in the k^{th} configuration, where $j \neq k$. Moreover, as aforementioned, there can be different numbers of clusters (i.e., workload labels) for each configuration. Therefore, to predict $WL_k(t+1)$ for $Config_k(t+1)$ using Markov chain, we need to know about its current workload $WL_k(t)$. Since at runtime, we only know about $WL_j(t)$ where $j \neq k$, we need to define the translation to the workload labels of other configurations. Thus, in order to find the best matching workload in configuration k with N_{wl_k} workload types (number of labels), we use the squared Euclidean distance metric (D_{Euc^2}), as follows:

$$wl_k^* \leftarrow \underset{wl_k^{'}}{\textbf{argmin}}(D_{Euc^2}(PC_{centroid}(WL_j(t)), PC_{centroid}(wl_k^{'})) \tag{1}$$
$$wl_k^{'} = \{1, ..., N_{wl_k}\}$$

where wl_k^* is the workload type of configuration k which is well matched to the current workload of configuration j, $WL_j(t)$. In this formulation, $PC_{centroid}(WL_j(t))$ shows the performance counter values related to cluster centroid of the workload at $Config_j(t)$, and $wl_k^{'}$ represents all known workloads for configuration k. Finally, we find the inter-configuration workload matches offline, and refer to them as a look-up table during runtime.

D. Performance Counter Estimator and Regression Model

Once the next workload for each configuration is predicted ($WL_k^{'}(t+1)$), the future throughput can be predicted through a regression model. As discussed earlier, each workload label corresponds to a cluster centroid composed of a set of performance counters. As shown in Section III-A, we can find a set of performance counters correlated with the throughput with smaller correlation coefficients between them. These counters are suitable candidates to provide a regression model to estimate the throughput. In this work, we consider Random Forest [17] with 50 weak learners for the regression algorithm, as it is able to sufficiently handle non-linear dependencies. We train an ensemble of bagged regression trees with this set of performance counters as the input features, and the corresponding throughputs, as the response variable.

Nevertheless, in the real-time execution, only a unique set of performance counter values can be extracted from $WL_k^{'}(t+1)$. Using these values into the trained regression model may cause a large prediction error, as the actual performance counter values may not be close enough to the centroids. Therefore, in order to compensate this error, we propose to use the ratio of actual performance counter values at time t and the estimated ones at time t by the centroids as a scaling factor for $PC_k(t+1)$, as follows:

$$\begin{aligned} PC_k^{'}(t+1) = PC_{centroid}(WL_k(t+1)) \odot \\ \big(PC_j(t) \oslash PC_{centroid}(WL_j(t))\big), \end{aligned} \tag{2}$$

where $PC_{centroid}(WL_k(t+1))$ is the performance counter values of the centroid that corresponds to the workload at time $t+1$ for the k^{th} configuration, and $\odot$ and $\oslash$ represent the element-wise product and division operations, respectively. Finally, the estimated performance counters, $PC_k^{'}(t+1)$, can be used in the trained regression model to predict the next throughput, $TH_k(t+1)$, for all available future configurations, $Config_k(t+1)$.

IV. Case Study: Real-time High Efficiency Video Coding (HEVC)

High Efficiency Video Coding (HEVC) is a next-generation video coding standard with doubled quality as its predecessors, H.264, for the same bitrate, at the cost of increased computational complexity [19]. This increased computational complexity makes real-time power-efficient video encoding very challenging. Nonetheless, HEVC is equipped with powerful tools that enable multi-threaded processing. Such features make multicore platforms promising to achieve real-time video encoding through workload parallelization. In this context, the real-time encoding is processing a particular number of frames every second, usually denoted as frame rate and measured in Frames Per Second (FPS). In this work, we consider a frame rate of 24 FPS, which is the most common in state of the art. However, the functionality of our proposed framework does not depend on the target throughput.

Real-time HEVC encoding is a highly time-varying application with iterative nature, i.e., it goes through different phases of the application several times as shown in Fig. 3 (as a video is composed of hundreds or thousands of frames). This nature of the video encoder makes the workload prediction and throughput estimation a promising technique for power and performance management. The main difference of HEVC encoding from other applications with an iterative structure is the rapid variation of the workload in each iteration of the encoder, as the frame contents within a video vary [20].

Real-time power efficient HEVC encoding has recently attracted a lot of attention [21], [22]. In particular, one of the most recent works [22] proposed a multi-agent reinforcement learning solution. However, such an approach requires application-specific hyper-parameter tuning to achieve the optimal performance. Thus, it cannot be used as a comprehensive framework for any arbitrary application.

V. Experimental Setup

We perform our experiments on a server equipped with two Intel Xeon E5 v4 CPUs providing a total number of 32 threads. Our server is able to perform DVFS with 16 available frequencies ranging from 1.2 GHZ to 3.6 GHz. Therefore, there are $32 \times 16 = 512$ different system configurations available for running each application. We use the Linux Perf tool [23] to collect the performance counters data. In order to provide accurate data about the application behavior, we use Docker containers to isolate the application from any possible background tasks.

Depending on the target frame rate, some of the system configurations can never satisfy the real-time HEVC encoder requirements, while some others may result in excessive throughput which increases power consumption and require larger buffers. Furthermore, if trained with all spare configurations that can never provide a satisfactory power consumption with the target throughput, the accuracy of the workload prediction and throughput estimation framework can degrade considerably. Hence, for our target case-study application, based on our observation when collecting training data for our framework, we narrow down the available configurations to 192 configurations. This selection is obtained by assuming two threshold values for the output framerate. In particular, in this work, we considered configurations that are able to reach at least 18 FPS once in the whole execution and do not provide an average framerate of larger than 35 FPS. We experimentally found this boundary around the target framerate (24 FPS) sufficient to account for any content variation within a video and between different videos.

VI. Experimental Results and Discussion

In this section, we first evaluate the accuracy and efficiency of each block in the proposed framework w.r.t. our case-study application. Afterwards, we provide a comparison with a neural network-based approach to assess the throughput estimation accuracy when dealing with a power minimization problem under QoS constraint. In particular, we adapt [3] for the design space of our target multi-core server. Since this work similar to ours, only relies on performance counters, we can directly adapt it to our own setup (CPUs instead of GPUs).

A. Performance and Accuracy of the Proposed Framework

1) Counter Selection: TABLE I lists the Pearson correlation matrix with one digit precision for the main counters and the

TABLE I
PEARSON CORRELATION MATRIX OF DIFFERENT PERFORMANCE COUNTERS AND HEVC ENCODER THROUGHPUT (FPS)

	branch miss	**bus cycle**	**L2 miss**	**floating point instr.**	**total Instr.**	**FPS**
branch miss	1.0	0.0	0.0	0.6	0.0	0.1
bus cycle	0.0	1.0	0.0	0.7	0.9	0.8
L2 miss	0.0	0.0	1.0	0.0	0.0	0.0
floating point instr.	0.6	0.7	0.0	1.0	0.8	0.8
total Instr.	0.0	0.9	0.0	0.8	1.0	0.9
FPS	0.1	0.8	0.0	0.8	0.9	1.0

throughput, FPS. On one hand, the total number of retired instructions (*total Instr.*) has the highest correlation with the output FPS. The next two hardware events with the largest correlation coefficient w.r.t. the throughput are, respectively, floating point arithmetic retired instructions (*floating point Instr.*) and bus cycles. On the other hand, *total Instr.* is highly correlated with the other two events, while *bus cycles* and *floating point Instr.* are less correlated. Hence, we consider only *floating point Instr.* and *bus cycles* which can well model the FPS through regression. On the contrary, for the workload clustering and classification, we consider all the listed counters, except for *total Instr.*, as each of the rest can demonstrate a particular aspect of the workload.

2) Per-Configuration Workload Clustering and Classification: As discussed in Section III-B, for each configuration, a different optimal number of clusters (workload labels) can be found. For our case-study application, we found 128 configurations with 2, 59 configurations with 3, and 5 configurations with 4 workload labels. Our study shows that, within those configurations providing higher throughputs more workload types can be found. Subsequently, Fig. 4 shows the histogram of the workload classification accuracy for the system configuration based on the observed event counters. On average, our classification achieves an accuracy higher than 99%.

3) Per-Configuration Workload Prediction: Fig. 5 shows the histogram of the workload prediction accuracy for the system configuration. While the average accuracy is 87%, lower accuracy values (e.g., around 70%) are mainly due to the fact that for some configurations with very large throughput, there is a rapid traverse from one content to a completely different one (and hence, different workload). Nonetheless, in practice, since these particular configurations often provide very large unnecessary throughput and, thus, higher power consumption, they are not proper candidates when solving the power minimization problem. Hence, as shown in Section VI-B, the overall results are only slightly affected.

4) Throughput Regression: The regression accuracy is computed across all configurations in terms of Root Mean Square Error (RMSE). In our case of study, the RMSE is 0.54 FPS.

B. Throughput Estimation Accuracy and Evaluation for Power Minimization

As explained earlier, the proposed framework provides throughput estimates for different available system configurations. In order to evaluate the accuracy of these estimates, we consider a power minimization problem in which system configuration changes dynamically to satisfy the required QoS

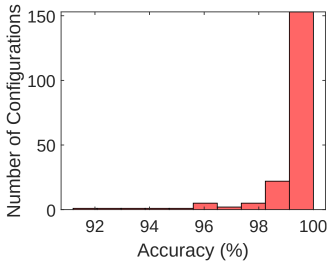


Fig. 4. Classification accuracy

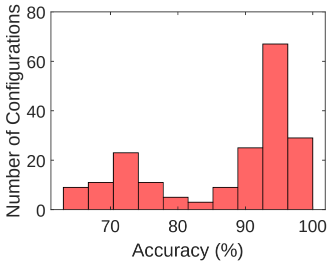


Fig. 5. Prediction accuracy

Algorithm 1: Power Minimization

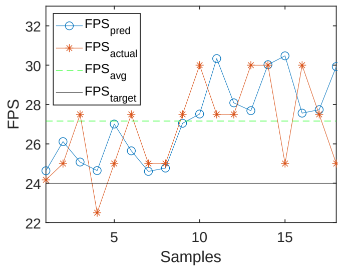
```
Input  : $TH^{'}(Config_k(t+1))$
Output: $Config_{opt}(t+1)$
1 $\boldsymbol{Config_{candidate}} \leftarrow$ Find$(Config_k(t+1))$
2     $s.t.$   $TH^{'}(Config_k(t+1)) \geq TH_{const}$
3 $Config_{opt}(t+1) \leftarrow \underset{Config_{k'}}{\textbf{argmin}} \; (P_{RF}(Config_{k'}),$
          $Config_{k'} \in \boldsymbol{Config_{candidate}}$
```

while providing the minimum power consumption. For this purpose, we provide a general power model. In particular, we run PARSEC 3.0 benchmark suit [24] with native input size under all available configurations and measure the average power consumption of package and DRAM. Then, we train an ensemble of bagged regression trees of RF with 50 weak learners. We consider the system configurations as the input features and the measured power consumption as the response variable. We test our trained ensemble with measured power consumption of HEVC encoder under different configurations. Our model provides an average error of 5.3% with standard deviation of 3.9%

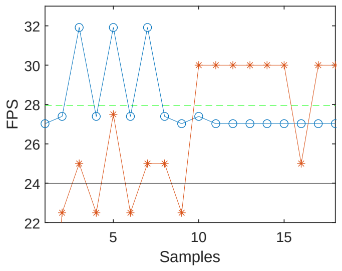
Using the obtained power model, we apply Algorithm 1 to the output of our proposed framework and that of [3]. The input to this algorithm is the estimated throughput, $TH^{'}(Config_k(t+1))$, for all available configurations. Here, the goal is to find a configuration by which the minimum power consumption can be achieved, while satisfying the throughout constraint, TH_{const}. In this algorithm, $\boldsymbol{Config_{candidate}}$ shows a vector of all configurations which are predicted to achieve a throughput larger than the constraint (Line 1 and 2). We use the obtained power model from the RF algorithm (P_{RF}) to estimate the power consumption of each candidate configuration. Finally, the one that achieves the minimum power consumption is used to run the application for the next time slot (Line 3).

Fig. 6(a) shows the throughput estimation obtained from our framework, the actual throughput, the average throughput, and the target throughput for a particular runtime window. Then, Fig. 6(b) shows the same plots as in Fig. 6(a) for the same test video, but as obtained from [3]. As shown in these figures, our framework is able to provide more accurate throughput estimation. For the test case shown in these figures, our framework is able to predict the output framerate with only 6.8% error, while this error for [3] is 20.5%. Furthermore, in the shown window, our framework violates the target throughput only once, while [3] shows four QoS violations.

As shown in Fig. 6(a), our framework is able to provide very



(a) Our framework



(b) State of the art [3]

Fig. 6. Throughput prediction and actual throughput

TABLE II
AVERAGE THROUGHPUT ESTIMATION ERROR AND QOS VIOLATIONS

	Min.(%)	**Max.(%)**	**Mean(%)**	**STD(%)**	**#QoS Violations**
Proposed	0.0	26.1	7.5	6.1	18
SoA [3]	14.4	30.0	20.1	5.5	61

accurate estimates for some samples, while for some others the estimation error is larger (e.g., 25% as for sample 15). The main source of such an error is the *Workload Prediction* rather than the *Workload Classification*, since as shown in Section VI-A2, the latter is highly accurate. In particular, such a large error can occur if the selected configuration for the next time slot is the one for which the average workload prediction accuracy is not sufficiently high. For this specific sample, the selected configuration is $(1.5GHz, 10threads)$with a workload prediction accuracy of 70%, on average.

In addition, we perform a K-fold cross-validation analysis. In contrast to our proposed framework, the work of [3] is more sensitive to the training data. The reason lies in the fact that [3] trains its neural network with the average throughput of each configuration. Consequently, it performs more accurately if the test data (i.e., video in our case) comes with almost the same variation range compared to the training data. However, this is not the case for HEVC and other highly time-varying applications. In particular, TABLE II shows the average throughput estimation error and QoS violations. On average, for different folds considered in our sensitivity analysis, our framework violates the target throughput 3.4x less than [3].

Finally, our proposed framework is able to save 15% more power consumption than the proposed approach in [3]. This additional power saving is due to the more accurate throughput estimation obtained from our framework. In fact, as shown in Fig. 6(b), there are several points where [3] underestimates the throughput and has to select a configuration with either higher frequency or larger number of threads than required, which leads to higher power consumption. In particular, from Sample 11 to Sample 15, [3] chooses 1.6 GHz as the operating frequency of 10 threads, and is unable to find any other configuration to satisfy the QoS with a lower power consumption. Nevertheless, 1.5 GHz and 9 threads as system configuration could also satisfy the QoS requirement with less power consumption. Unfortunately, [3] discards this configuration because it mispredicts the corresponding throughput to be less than 24 FPS (see Line 1 in Algorithm 1).

VII. Conclusion

In this work, we have proposed a machine learning-based framework for proactive power and performance management of time-varying applications. We compared the proposed framework with the state of the art considering real-time HEVC encoder as a case-study which represents a QoS-sensitive application with iterative structure and large workload variations. Our results showed that our framework improves QoS violations by 3.4x, while decreasing power consumption by 15% due to its more accurate predictions than state-of-the-art approaches.

References

[1] J. Flich *et al.*, "Mango: Exploring manycore architectures for next-generation hpc systems," in *DSD*. IEEE, 2017.
[2] I. Intel, "and ia-32 architectures software developers manual," *Volume 3A: System Programming Guide, Part*, vol. 1, no. 64, p. 64, 64.
[3] G. Wu *et al.*, "GPGPU performance and power estimation using machine learning," in *HPCA*. IEEE, 2015.
[4] V. Podolskiy *et al.*, "Forecasting models for self-adaptive cloud applications: A comparative study," in *SASO*. IEEE, 2018.
[5] R. N. Calheiros *et al.*, "Workload prediction using arima model and its impact on cloud applications qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.
[6] D. Yi *et al.*, "New driver workload prediction using clustering-aided approaches," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 1, pp. 64–70, 2019.
[7] Y. Tan *et al.*, "Workload prediction and dynamic voltage scaling for mpeg decoding," in *ASP-DAC*. IEEE, 2006.
[8] S. Browne *et al.*, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," in *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. IEEE, 2000.
[9] X. Zheng *et al.*, "Accurate phase-level cross-platform power and performance estimation," in *DAC*. IEEE, 2016.
[10] A. Borghesi *et al.*, "Predictive modeling for job power consumption in hpc systems," in *HiPC*. Springer, 2016.
[11] K. Singh *et al.*, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.
[12] M. Curtis-Maury *et al.*, "Online power-performance adaptation of multithreaded programs using hardware event-based prediction," in *ICS*. ACM, 2006.
[13] A. C. De Melo, "The new linuxperftools," in *Slides from Linux Kongress*, vol. 18, 2010.
[14] J. Benesty *et al.*, "Pearson correlation coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.
[15] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.
[16] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.
[17] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[18] R. G. Gallager, *Stochastic processes: theory for applications*. Cambridge University Press, 2013.
[19] G. J. Sullivan *et al.*, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.
[20] A. Iranfar *et al.*, "Machine learning-based quality-aware power and thermal management of multistream HEVC encoding on multicore servers," *IEEE TPDS*, vol. 29, no. 10, 2018.
[21] ——, "Online efficient bio-medical video transcoding on MPSoCs through content-aware workload allocation," in *DATE*. IEEE, 2018.
[22] L. Costero *et al.*, "Multi-agent reinforcement learning for efficient real-time multi-user video transcoding," in *DATE*, 2019.
[23] A. C. De Melo, "The new linuxperftools," in *Slides from Linux Kongress*, vol. 18, 2010.
[24] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, 2011.