Phase-Aware CPU Workload Forecasting

Erika S. Alcorta[1], Pranav Rama[1], Aswin Ramachandran[2], and Andreas Gerstlauer[1]

[1] The University of Texas at Austin, Austin TX, USA
{esalcort, pranavrama9999, gerstl}@utexas.edu
[2] Intel Corporation, Austin TX, USA
aswin.ramachandran@intel.com

Abstract. Predicting workload behavior during execution is essential for dynamic resource optimization of processor systems. Early studies used simple prediction algorithms such as a history tables. More recently, researchers have applied advanced machine learning regression techniques. Workload prediction can be cast as a time series forecasting problem. Time series forecasting is an active research area with recent advances that have not been studied in the context of workload prediction. In this paper, we first perform a comparative study of representative time series forecasting techniques to predict the dynamic workload of applications running on a CPU. We adapt state-of-the-art matrix profile and dynamic linear models (DLMs) not previously applied to workload prediction and compare them against traditional SVM and LSTM models that have been popular for handling non-stationary data. We find that all time series forecasting models struggle to predict abrupt workload changes. These changes occur because workloads go through phases, where prior work has studied workload phase detection, classification and prediction. We propose a novel approach that combines time series forecasting with phase prediction. We process each phase as a separate time series and train one forecasting model per phase. At runtime, forecasts from phase-specific models are selected and combined based on the predicted phase behavior. We apply our approach to forecasting of SPEC workloads running on a state-of-the-art Intel machine. Our results show that an LSTM-based phase-aware predictor can forecast workload CPI with less than 8% mean absolute error while reducing CPI error by more than 12% on average compared to a non-phase-aware approach.

Keywords: Run time workload prediction · time series forecasting.

1 Introduction

Predicting dynamic workload behaviors has become an essential step in optimizing hardware resources at runtime. For example, anticipating an application's memory intense period can result in power savings if the power management module switches the core frequency promptly. In addition to power management, prediction of workload metrics such as CPI has also been exploited in a variety of applications including reduction of task interference in multi-tenant systems [13], task migration and scheduling [19], and defending against side-channel attacks [17]. Predictions allow systems to behave proactively instead of reactively. It has been previously shown that proactive decisions yield better optimization results [1]. However, proactive approaches are challenging because they require predicting the future.

Looking at the past is often a reliable way of estimating the future. Program applications specifically present variable workload behaviors throughout their execution and many of them exhibit periodic trends or patterns. Workload prediction techniques exploit these characteristics to estimate future behaviors. Early work in dynamic workload forecasting investigated basic methods such as exponential averaging and history tables [7]. Later studies proposed more advanced approaches, ranging from linear regression [20] to, more recently, recurrent neural networks (RNNs) [13]. Their objective is to minimize the forecasting error of periodically measured CPU workload metrics, such as CPI. This periodic collection of metrics forms a time series; hence, runtime workload behavior forecasting is formally a time series forecasting problem [6, 7, 20, 26].

Time series analysis has been studied for numerous applications, such as stock price prediction, earthquake detection and traffic forecasting [15]. Researchers have proposed many recent advances in these fields that have not been studied in the context of dynamic workload forecasting. In this paper, we first perform a comparative study of representative time series forecasting models applied to predicting CPU workload metrics on a single core. We focus on models that can handle non-stationary program behaviors. We compare classic support vector machine (SVM) [21] and RNN-based long-short term memory (LSTM) [8] regressors against auto-regressive dynamic linear models (DLMs) [11] from the controls domain as well as predictors based on state-of-the-art matrix profile (MP) [27] time series data mining models.

Our results show that all time series forecasting techniques struggle to predict abrupt workload changes. Such changes occur because workloads go through phases. Program phases and their detection, classification and prediction at runtime have been extensively studied [4]. Phase predictors excel at predicting abrupt workload changes since, by definition, a phase is composed of intervals of execution with similar behaviors. A change of phase is thus a change in average workload behavior. We propose to complement time series forecasting with phase classification and prediction. Our approach trains multiple regression models, one per program phase. At runtime, sampled workload traces are fed into the appropriate phase-specific model and forecasted workload metrics are selected and concatenated based on the output of a phase classifier and predictor. Our results show that complementing time series forecasting with phase prediction consistently decreases the forecasting error of all forecasting techniques and programs that go through phases.

We summarize the contributions of this paper as follows:

1. We perform a comparative study of representative time series forecasting techniques to predict application workload behavior at run time. Our com-

parative study includes state-of-the-art time series techniques that, to the best of our knowledge, have not previously been adopted for time series forecasting before.
2. We propose to complement time series forecasting techniques with phase prediction by implementing a separate forecaster per workload phase, which results in significant reductions of forecasting errors for all benchmarks.
3. We perform our study and evaluate our approach for prediction of large-scale SPEC benchmark behavior running on state-of-the-art CPUs for up to 20 time steps into the future. Results show that a phase-aware LSTM provides the best predictions, where a phase-aware approach improves prediction accuracy by more than 12% compared to a non-phase-aware setup.

The remainder of this paper is organized as follows. We review the related work in the next section. In Section 3, we provide background about the forecasting models that we evaluate. We summarize the workload forecasting formulation and explain how we complement it with phase prediction in Section 4. Section 5 presents the experimental methodology and Section 6 shows our results. Finally, we present our conclusions in Section 7.

2 Related Work

Time series analysis applications are prevalent in economics, demography, industrial process control, etc. [15]. Time series forecasting has been used in a wide range of computing applications as well. In [3], the auto-regressive moving average (ARMA) model was compared against exponential averaging, history table predictor, and least squares regression for thermal prediction in multiprocessor SoCs. In data centers, forecasting has been used to predict cluster utilization [24]. Nikravesh et al. [16] noticed that SVMs and MLPs have comparative accuracies in predicting data center user requests over time. Matrix profile is a state-of-the-art technique used for time series motif discovery and analysis [27]. It has been applied in detecting anomalies in CPU utilization traces of various workloads [5]. However, existing work has not studied matrix profile for time series prediction. In our work, we specifically demonstrate its adoption for workload forecasting.

Early studies in forecasting dynamic workload metrics proposed basic statistical and table-based predictors. Duesterwald et al. [7] compared a last-value predictor with exponentially weighted moving average (EWMA) and history predictors to forecast instructions per cycle (IPC) and L1D cache misses. The history table predictor resulted in the lowest mean absolute error (MAE). Another study [20] evaluated linear regression to forecast IPC. The results show that they have a lower MAE than the last-value predictor. Kalman filters have been recently used in the context of CPU workload prediction [14]. They are used to predict cycles per instruction (CPI) to optimize dynamic energy management. One of the forecasting models that we study in this work is DLM [11], which uses a state-space representation similar to Kalman filters. It additionally can capture short-term periodicity and trends in time series, but has not been applied to workload prediction before. Advanced machine learning techniques have shown to be more accurate than traditional predictors. Zaman et al. [26] found that a SVM regressor results in the lowest MAE when forecasting various performance counters. They compared an SVM against last-value, history table, and ARMA predictors. ARMA is an auto-regressive predictor that assumes that the time series is stationary. Since workload behaviors are not stationary, we include DLM as representative auto-regressive technique that does not assume stationarity. With recent popularity of RNNs, a later study [13] investigated the design space of LSTMs to forecast IPC and other metrics of workloads when they are co-allocated with other tasks in data centers. They compared LSTMs against linear regression and MLPs, concluding that LSTMs result in the highest coefficient of determination (R^2) scores.

Multiple studies have proposed to detect and classify workload phases using hardware counters. Early work [9] categorized the memory-boundedness of a workload into phases. A more recent study [10] uses unsupervised learning to cluster samples of hardware counters. In addition to detection and classification, studies in phase prediction focus on predicting discrete workload transitions, i.e., its phase changes. In [9], a global phase history table was proposed. In [10] a genetic algorithm uses phase labels and other parameters for thermal prediction, where changes occur at a slower pace as opposed to CPU workload prediction, where changes can be abrupt. Laun et al. [12] compared Markov tables with last-value predictors. They observed that the same phase is detected in consecutive sampling periods and proposed to use run-length encoding to predict phase changes and estimate phase duration interval ranges. This observation has been made in more recent studies as well. Srinivasan et al. [23] proposed a linear adaptive filter to predict the duration of classified phases. To the best of our knowledge, however, there is no existing work that has short-term workload time series forecasting with phase prediction to capture long-term patterns. In this work, we aim to evaluate how the notion of phases impacts forecasting accuracy orthogonal to any specific phase prediction approach. As such, we implement an oracle predictor and leave research into phase predictors for forecasting to future work.

3 Background

This section describes in further detail the models that we evaluate in this study.

Support Vector Machines (SVMs) An SVM is a supervised learning model whose objective is to minimize an error bound instead of minimizing residuals. This objective has the purpose of generalizing unseen data [21]. We use SVMs for regression, which is commonly referred to as support vector regression (SVR). It is common to apply non-linear transformations, called kernels, to the SVM's input space. In this work, we show the performance of both linear and kernel SVRs. We use a radial-basis function (RBF) as kernel, which is expected to improve accuracy compared to a linear SVR at the expense of computational cost. SVMs take a vector of features as their input. Thus, we convert the multivariate history window to a single-dimensional space. In our study, when the forecast horizon, k, is greater than 1, multiple SVM models are learned independently.

Dynamic Linear Models (DLMs) Dynamic linear models (DLMs) [11] are recursive models formulated as state space models with state parameters corresponding to the structure of the time series. We include DLM components for the general trend of the time series, seasonality of a given size (to capture periodicity) and dynamic regression with predictor variables. These components are combined into state space form to iteratively estimate the next step in time series given the previous inputs of a certain window size. Due to the iterative nature, the model can only consider the previous input window to make a prediction. Making the window size and seasonality too large might result in infeasible computation time. As such, this model is suitable for short term but not long term periodicity.

Long-Short Term Memory (LSTM) LSTM networks are a type of RNN whose structure is characterized by having a memory unit that holds long-term information. The architectures used in this work is composed out of one or more stacked LSTM layers and one fully connected layer. The LSTM layers process the inputs in the time domain to encode a feature vector that the fully connected layer uses to output the forecasts. This architecture is formally classified as an acceptor LSTM. The fully connected layer has k output neurons, which simultaneously predict each value of the forecast horizon.

Matrix Profile (MP) Matrix profile is a recent and fast algorithm for uni-variate time series motif discovery [27]. Motifs are defined as pairs of subsequences of the same time series that are very similar to each other. We propose to adopt matrix profile for workload forecasting by finding a window in a workload time series y_t that is most similar to the most recent window of size h, $(y_{t-h+1}, ..., y_t)$. The samples that follow the most similar window are then used as the forecast values at time t. In other words, there is a subsequence $(y_{v-h+1}, ..., y_v)$, $v + h < t$, that matrix profile finds to be the most similar to $(y_{t-h+1}, ..., y_t)$. The k samples that follow v are then used as the forecast at time t, i.e. $(\hat{y}_{t+1}, ..., \hat{y}_{t+k}) = (y_{v+1}, ...y_{v+k})$

4 CPU Workload Forecasting

In the following, we first summarize the task of forecasting CPU workload metrics as used in this work. We then describe our proposal to combine forecasting with phase detection and prediction.

4.1 Basic Forecasting

Workload time series are formed from hardware counter data collected using the CPU's performance monitoring units (PMU). Multiple PMU counters are

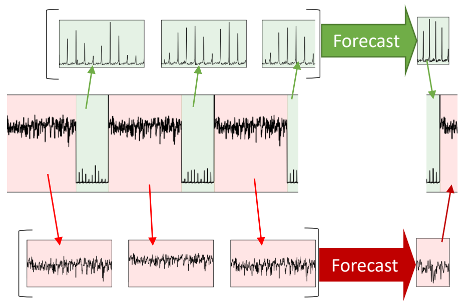


Fig. 1: Example of forecasting executions of two different phases of *nab*

collected each period, resulting in a multivariate time series. The forecasting techniques focus on predicting one of the counters and may use the rest of them as inputs for additional information. We run different programs and consider the execution of each program as a separate time series forecasting task.

Formally, each multivariate time series $U \in \mathbb{R}^{n\times m}$ is composed out of n observations of m variables. In the case of workload metric forecasting, the value of m depends on the maximum number of PMU counters that can be collected at the same time. We are interested in predicting one of the m variables, $y \in U$. The observation of this variable at time t, $1 \leq t \leq n$, is denoted as y_t. We are interested in predicting k future values of y at time t, $(\hat{y}_t + 1, ..., \hat{y}_{t+k})$, using only past observations $U_i, i \leq t$. k is known as the forecast horizon. At each step t, a predictor generally takes as input a history window of size h, $(U_{t-h+1}, ..., U_t)$. In summary, the time series forecasting problem can be formalized as follows:

$$(\hat{y}_{t+1}, ..., \hat{y}_{t+k}) = m_{p,w}\big((U_{t-h+1}, ..., U_t)\big), \qquad (1)$$

where $m_{p,w}$ represents the trained model function of predictor p with its set of trainable parameters w.

Finding model parameters w is a supervised learning problem. We use a subset of observations t in U with known true values $\hat{y}$ to create a training data set. During prediction at runtime, we use sliding windows of h history values for every new time step t in the test set to predict k future values rooted at t.

4.2 Phase-Aware Forecasting

As our results will show, basic forecasting techniques show low prediction accuracy for workloads that exhibit distinct long-term phase behavior, even when those phases repeat over time. We propose to alleviate this problem by expanding the scope of time series forecasting using phase detection and prediction. Figure 1 shows the intuition behind our approach. The center of the figure shows a snippet of the *nab* workload going through two different phases, highlighted as red and green regions. We partition the trace based on phases, concatenate the subtraces, and train a predictor specific to each phase. The forecasts belonging to a

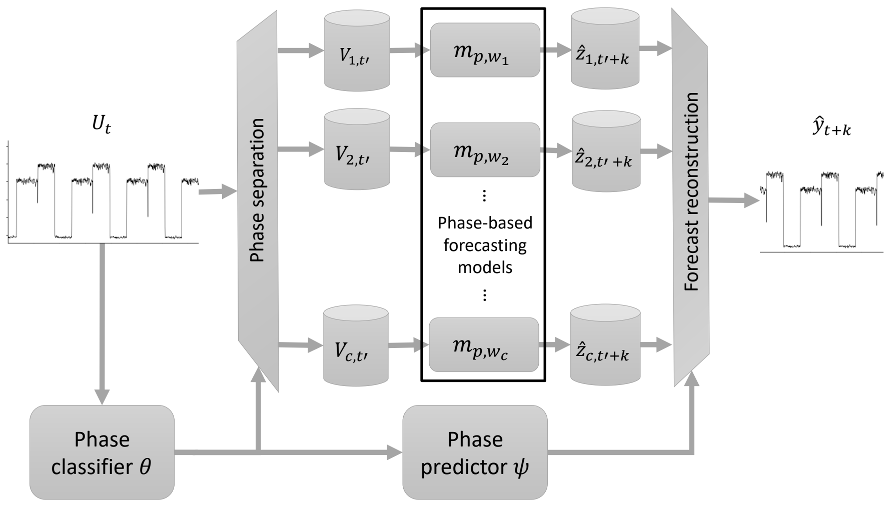


Fig. 2: Phase-aware workload forecasting.

phase are thus only dependent on the history of that phase, and phase-specific predictors can be specialized to a single phase to increase accuracy. Finally, with knowledge of the future phase behavior, phase-specific forecasts are concatenated and assembled to reconstruct the forecast for the overall time series.

Formally, we use c to denote the total number of distinct phases that a workload cycles through. An overview of our phase-aware forecasting approach is shown in Figure 2. Our approach consists of four high-level stages: (1) phase classification and separation, (2) phase-based forecasting, (3) phase prediction, and (4) forecast reconstruction. In the following, we formalize each step in detail.

Phase Classification and Separation A phase classifier, Θ, maps each sample, U_t, to a phase, α_t, $1 \leq \alpha_t \leq c$:

$$\alpha_t = \Theta(U_t). \tag{2}$$

The samples of U that share the same phase, i, are concatenated into a single vector. In total, there are c disjoint time series V_i, defined as follows:

$$V_i = \big(U_t | \alpha_t = i\big), 1 \leq i \leq c \tag{3}$$

with observations $V_{i,t'}$, where t' represents the mapping of original observations U_t into a new time dimension t' for each series. Note that the following conditions must be true:

$$U = \bigcup_{i=1}^{c} V_i, \text{and} \bigcap_{i=1}^{c} V_i = \emptyset. \tag{4}$$

Phase-Based Forecasting Each new time series, V_i is processed by a different model m_{p,w_i}, where p again denotes the prediction technique and w_i the corresponding set of trained parameters. Note that all models use the same model architecture and predictor type, i.e. they differ only in trained parameters. The models are otherwise handled in the same way as in Section 4.1; they use h samples to forecast k values of a predicted variable $z_i \in V_i$ as follows:

$$(\hat{z}_{i,t'+1}, ..., \hat{z}_{i,t'+k}) = m_{p,w_i}\big((V_{i,t'-h+1}, ..., V_{i,t'})\big). \tag{5}$$

Phase Prediction A phase predictor, ψ, further takes outputs α_t from the phase classifier to predict k future phases at time t based on the history of d previous phases. In other words:

$$(\hat{\alpha}_{t+1}, ..., \hat{\alpha}_{t+k}) = \psi\big((\alpha_{t-d+1}, ..., \alpha_t)\big). \tag{6}$$

Forecast Reconstruction Finally, since the forecasting models m_{p,w_i} are unaware of their interactions and relationships to the original time series, we use the outputs $\hat{\alpha}_t$ from the phase predictor to select those values of $\hat{z}_{i,t'}$ that should be output as overall forecast $\hat{y}_t$. Formally:

$$(\hat{y}_{t+1}, ..., \hat{y}_{t+k}) = (\hat{z}_{\hat{\alpha}_{t+1},t'+1}, ..., \hat{z}_{\hat{\alpha}_{t+k},t'+k}). \tag{7}$$

5 Experimental Methodology

To generate the workload traces, we use a subset of programs from the SPEC CPU 2017 benchmark suite [22]. The subset was chosen to represent different representative workload phase behaviors: a uniform pattern (*nab*), abrupt transitions (*cactuBSSN*), hard to predict non-stationary patterns (*mcf*), long phase durations (*xz*), and workloads with only a single phase (*perlbench*). We used the *ref* inputs given by the benchmark suite. The execution time of all workloads is more than one minute, which provides enough data to train the forecasting models. Table 1 summarizes the workloads and their phase characteristics.

Our target platform is an Intel Xeon-SP running Ubuntu 18.04. We collect PMU counters every 10ms using Intel's EMON command-line tool. We constrained our data features to the number of PMU events that can be accessed simultaneously. In the case of the Intel Xeon-SP platform, 4 fixed counters and up to 8 variable counters can be sampled per core (or uncore) simultaneously. In addition to fixed instruction and cycle counters, we selected 8 counters for prediction that can characterize the workload behaviors by their memory boundedness (L2 accesses, L2 hits and L3 misses, i.e. main memory accesses), control flow predictability (retired total and mispredicted branch instructions), and operation mix (retired floating point operations). We also use executed μOps and stall cycles to account for other resources stalling the CPU pipeline execution. Normalizing the PMU counters to the number of instructions yielded more accurate results for all multi-variate models. We also found that reducing dimensionality

Table 1: Benchmark summary.

Benchmark	Samples	No. of phases	Avg. phase length
cactuBSSN	202,179	5	167
mcf	52,673	5	599
nab	170,251	5	231
perlbench	16,462	1	—
xz	126,669	4	7,037

with principal component analysis (PCA) improves the performance of SVR and LSTM models. Finally, we applied a median filter to the data to eliminate noise. A median filter was preferred over other smoothing techniques for its ability to preserve workload behavior changes.

CPI is used as the variable of interest, y, to compare our models. The rest of the collected performance counters are used by some of the models as inputs. The matrix profile algorithm is designed for uni-variate time series; therefore, we only use CPI as input. The rest of the models use the other counters in a multi-variate fashion as described in Equation (1).

We split each time series into 70% of samples used for training, hyperparameter tuning and model selection, and 30% of samples for testing. We use the mean absolute percentage error (MAPE) between measured and predicted CPI to evaluate our forecasting models. We set a fixed forecast horizon of all models of $k = 20$. With this, we compute a separate $MAPE_i$ for every step $1 \leq i \leq k$ in the forecast horizon as follows:

$$MAPE_i = \frac{100\%}{n} \sum_{t=1}^{n-k} \left(\frac{|y_{t+i} - \hat{y}_{t+i}|}{y_{t+i}} \right) \tag{8}$$

When comparing different models, we look at the average MAPE ($AMAPE$) of all 20 predictions: $AMAPE = \frac{1}{k} \sum_{i=1}^{k} MAPE_i$.

In addition to evaluating forecast errors, we measured the inference time corresponding to one prediction step. To perform a fair comparative study, we run the inference of all forecasting models on the same machine, an Intel Core i9-9900K running Debian 9.13. We use Python's *time* standard library to measure the inference time. The frameworks that we use to implement and train our forecasting models are *PyDLM* [25] for DLM, *scikit-learn* [18] for SVM, *Keras* [2] for LSTM, and *PySCAMP* [28] for matrix profile.

To evaluate the benefits of phase-aware forecasting independent of a specific phase prediction approach that comes with its own inaccuracies, we use an oracle phase predictor. We focus our work on the study of forecasting models in phase-aware versus -unaware settings, and leave the selection of phase predictors to future work.

6 Experimental Results

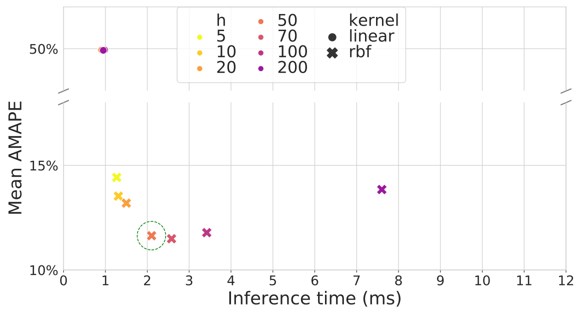
We first discuss tuning of hyperparameters and selection of forecasting model architectures. We then evaluate and compare accuracy of different phase-aware versus -unaware variants of each model. As described above, we use 70% of the trace of each benchmark for training and 30% for testing. For hyperparameter tuning and model selection, we further split the training set into 50% of samples used for exploration, i.e. to train the models with different hyperparameters, and 20% used for validation, i.e. to select the best parameters based on the average performance across benchmarks. We then train the final models with the complete 70% of samples and use the remaining 30% in the test set to evaluate accuracy of each forecasting technique.

6.1 Hyperparameter Tuning and Model Selection

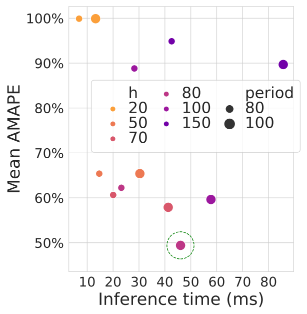
We evaluate each technique's sensitivity to history window size, h, and other relevant model parameters to select the best overall architecture for each model. Figure 3 shows the exploration of all forecasting models. We plot the tradeoff between the mean AMAPE across all benchmarks on the y-axis and the inference time on the x-axis. We also show the finally selected hyperparameters of each model with a dotted green circle on each figure.

For SVMs (Figure 3a), we explored both linear and RBF kernels. Given inherently non-linear workload behavior, RBF kernels show significantly better accuracy, but come at the expense of higher computational cost. Window size impacts inference time with an RBF kernel, where more complex models with larger input features increase computation time. By contrast, the forecast error is very similar across window sizes. In general, a window size that is too small to capture workload periodicity will result in larger errors. At the same time, very long window sizes result in a model that averages samples instead of learning their interactions. The optimal window size strongly depends on the workload, however. The mean AMAPE is lowest for a window size of 70, but smaller window sizes have better accuracy for *nab* while larger window sizes are better for *xz*. We chose a window size of $h = 50$ due to its faster inference time and mean AMAPE that is very close to $h = 70$ (less than 1% difference).

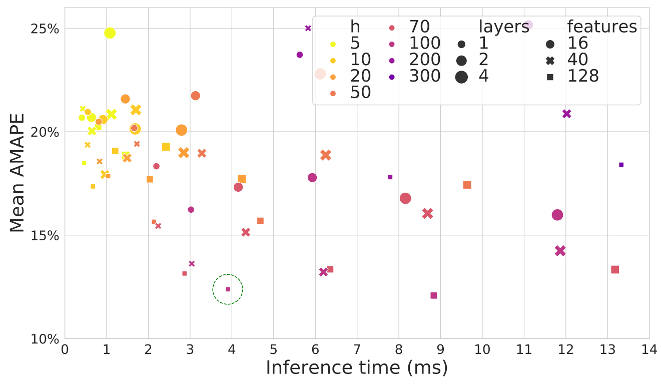
For DLM (Figure 3b), in addition to history window sizes, we selected seasonality (periodicity) through validation. Larger periodicity improves accuracy regardless of the window size, albeit at the cost of a significant increase in computation time. Window sizes show similar accuracy and inference time trends than in SVMs, but they have a stronger impact on accuracy for DLMs. Medium window sizes show best accuracy at intermediate computation costs. These trends in window sizes and periodicity were consistent in all benchmarks. The best mean AMPAE with reasonable computation time was found to be for a window size of $h = 80$ with periodicity of 100. Our DLM model also includes a degree 2 trend component and a multivariate dynamic regression component with 8 predictor variables. The DLM was found to do better with the multivariate predictor variable component compared to without it.



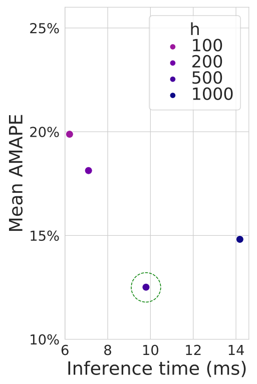
(a) SVM exploration



(b) DLM exploration



(c) LSTM exploration



(d) MP exploration

Fig. 3: Exploration of the forecasting techniques hyperparameter space.

The exploration of LSTM hyperparameters (Figure 3c) included the number of LSTM layers and the number of features per layer in addition to input window size h. They all impact the computational costs, with fewer features, fewer layers and smaller window sizes being faster. Having more features reduces the forecasting error. This is a trend we observed across all benchmarks except *perlbench*, which showed an opposite trend. We thus selected 128 as the number of features. A smaller number of layers generally decreases mean AMAPE, but the impact on forecasting error is dependent on the number of features and the workload. For example, for *nab*, increasing the number of layers with 16 features reduces AMAPE, but the trend is opposite with 40 features. The models with lowest mean AMAPE, however, all have 1 and 2 layers. As such, we chose 1 layer for our final model as the inference time is faster and the mean AMPE difference is not significant. The window sizes with lowest mean AMAPE were 70 and 100. The best performing window, however, was different for each benchmark. The best window size was 100 for *cactuBBSN*, 70 for *mcf*, 50 for *nab* and 10 for *xz*.

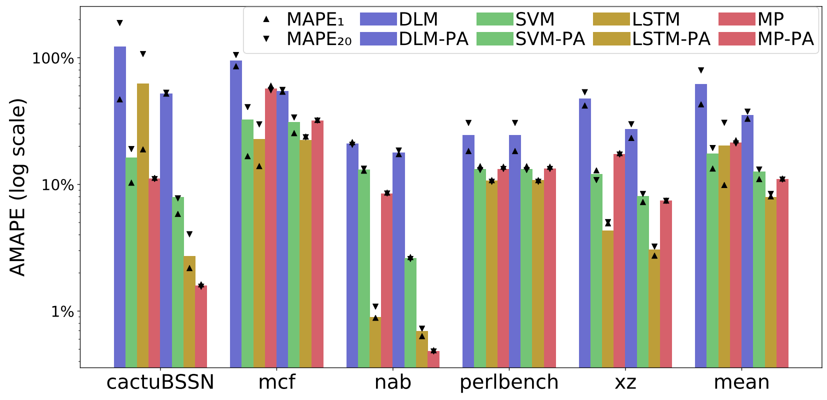


Fig. 4: Accuracy of phase-unaware versus -aware (PA) models.

Similar to SVM and DLM models, the error decreases up to those values and then increases again. The mean AMAPE is best for $h = 100$, which is what we chose in the end.

Finally, the range of window sizes that we show for matrix profile (Figure 3d) is at a larger scale than the other forecasting techniques. This is because MP did not perform well with smaller window sizes. Similar to other models, we observed that for most benchmarks, the forecast error decreases with increasing window sizes up to a certain value, while larger window sizes increase computational cost. The only benchmark that was not significantly impacted in accuracy by the window size was *xz*. The best mean AMAPE for matrix profile was for $h = 500$, which we chose for this model.

Overall, with the exception of DLM, all models show similar validation accuracies and inference times. DLM, however, is both significantly more inaccurate and slower than other approaches. The technique with the fastest inference time is SVM with 2.1ms, followed by LSTM with 3.9ms, matrix profile with 9.8ms and lastly, DLM with 46ms. These time measurements were taken with the purposes of comparing the inference times of different models relative to each other in their base software implementation. Further investigation is required to optimize their implementations and reduce overhead for actual deployment, e.g., by pruning or hardware acceleration. We evaluate final model accuracy for the test set in the next section.

6.2 Accuracy Evaluation

Figure 4 shows the accuracy comparison of all four forecasting techniques using a basic and our proposed phase-aware (PA) setup. In addition to $AMAPE$, the graph also shows the range of $MAPE_1$ and $MAPE_{20}$ across the nearest and farthest prediction in the forecast horizon for each model and benchmark. Most benchmarks and models show that the closest forecast is more accurate than the farthest, with the mean between them.

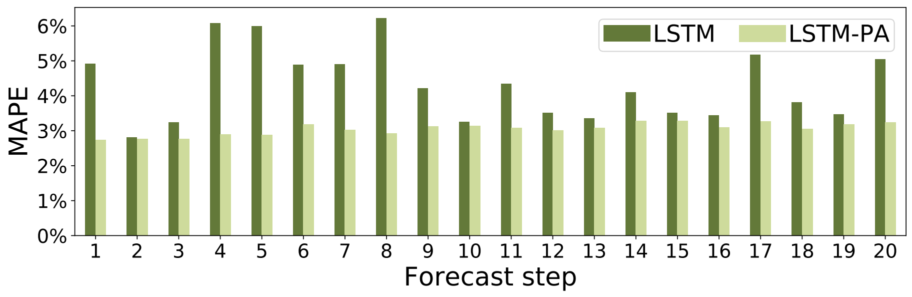


Fig. 5: Accuracy per forecast steps of phase-aware and -unaware LSTM for *xz*.

When comparing traditional models with phase-aware approaches, results show that using phase-specific models consistently decreases the forecasting error of all techniques for all benchmarks except *perlbench*. The workload traces of *perlbench* do not go through phases and there is no room of improvement for phase-aware forecasting.

We observe that *cactuBSSN* exhibits the most impact in forecasting error reduction with a phase-aware approach. Some of its transitions between phases have very abrupt changes where the CPI value increases by 500%. Any mispredictions of these transitions result in very large error penalization. This is also reflected in the large variation in errors between the first and last prediction in the forecast horizon. A phase-unaware LSTM in particular struggles to predict those changes and benefits significantly from a phase-aware approach. By contrast, *mcf* is impacted the least from phase-aware models and generally exhibits poor accuracy and larger error variations. This is because its phases continuously change and reduce in length over time, which makes the workload hard to predict overall. Note that matrix profile in particular cannot accurately predict this trend since its predictions are purely based on recalling past behavior unchanged. As opposed to *mcf*, the phases of *nab* have repetitive uniform patterns, where phase-aware models have a significant impact in decreasing forecast error. A basic LSTM is able to accurately learn both short-term and long-term phase patterns for this workload, but its phase-aware counterpart still had room for improvement.

Finally, Figure 5 shows the MAPE of all steps in the forecast horizon of a phase-aware and -unaware LSTM when predicting *xz*. While a phase-unaware LSTM provides good AMAPE across all steps, it shows high maximum errors due to its inability to predict phase changes with larger CPI jumps for this workload. The phase-unaware LSTM will sometimes predict a phase change when there is none or will fail to predict a change at the right time, which results in large errors for certain forecast steps. By contrast, the phase-aware LSMT shows small variations in errors across steps with a general trend of slightly increasing errors the further the predictions are made into the future.

7 Summary and Conclusions

In this paper, we formulated runtime CPU workload prediction as a time series forecasting problem and performed a comparative study among different representative techniques including classical auto-regressive (DLM), machine learning (SVM and LSTM), and a state-of-the-art motif discovery (matrix profile) approach that we proposed for workload forecasting. We showed that the main challenge in workload forecasting is the prediction of abrupt changes due to workload phase behavior. We proposed a novel phase-aware forecasting approach that leverages phase classification and prediction to separate time series into phases and train a separate, specialized prediction model for each phase. Results on a subset of SPEC 2017 benchmarks running on a state-of-the-art workstation show that phase-aware forecasting improves MAPE by 14% on average across different models and benchmarks. A phase-aware LSTM was the best performing predictor with less than 8% average MAPE across benchmarks and a forecast horizon of 20 steps. By contrast, a phase-aware SVM is almost twice as fast but at decreased accuracy of 13% MAPE. A phase-aware matrix profile predictor can in some cases outperform an LSTM, but at much higher computational cost.

Future work includes investigating phase-aware forecasting for a wider range of workloads, integrating phase predictors to complement phase-aware models, approaches for online training of predictors, efficient hardware or software deployment of predictors, application of phase-aware workload forecasting to various use cases such as power management or system scheduling, as well as workload forecasting for multi-threaded workloads running in multi-core settings, where task interference effects are considered in phase classification, detection and forecasting.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was supported in part by Intel and NSF grant CCF-1763848.

References

1. Ababei, C., Moghaddam, M.G.: A survey of prediction and classification techniques in multicore processor systems. IEEE TPDS **30**(5), 1184–1200 (2018)
2. Chollet, F., et al.: Keras. https://keras.io (2015)
3. Coskun, A.K., Rosing, T.S., Gross, K.C.: Utilizing predictors for efficient thermal management in multiprocessor SoCs. IEEE TCAD **28**(10), 1503–1516 (2009)
4. Criswell, K., Adegbija, T.: A survey of phase classification techniques for characterizing variable application behavior. IEEE TPDS **31**(1), 224–236 (2019)
5. Dieter De Paepe, O.J., Hoecke, S.V.: Eliminating noise in the matrix profile. In: ICPRAM (2019)
6. Dietrich, B., et al.: Time series characterization of gaming workload for runtime power management. IEEE TC **64**(1), 260–273 (2015)

7. Duesterwald, E., Cascaval, C., Dwarkadas, S.: Characterizing and predicting program behavior and its variability. In: PACT (2003)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
9. Isci, C., Contreras, G., Martonosi, M.: Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In: MICRO (2006)
10. Khanna, R., John, J., Rangarajan, T.: Phase-aware predictive thermal modeling for proactive load-balancing of compute clusters. In: ICEAC (2012)
11. Laine, M.: Introduction to Dynamic Linear Models for Time Series Analysis, pp. 139–156. Springer (2020)
12. Lau, J., Schoenmackers, S., Calder, B.: Transition phase classification and prediction. In: HPCA (2005)
13. Masouros, D., Xydis, S., Soudris, D.: Rusty: Runtime system predictability leveraging LSTM neural networks. IEEE CAL **18**(2), 103–106 (2019)
14. Moghaddam, M.G., Ababei, C.: Dynamic energy management for chip multiprocessors under performance constraints. Microprocessors and Microsystems **54**, 1–13 (2017)
15. Montgomery, D.C.: Introduction to Time Series Analysis and Forecasting. Wiley (2015)
16. Nikravesh, A.Y., Ajila, S.A., Lung, C.: Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In: SEAMS (2015)
17. Nomani, J., Szefer, J.: Predicting program phases and defending against side-channel attacks using hardware performance counters. In: HASP (2015)
18. Pedregosa, F., et al.: Scikit-learn: Machine learning in Python. JMLR **12**, 2825–2830 (2011)
19. Rapp, M., Pathania, A., Mitra, T., Henkel, J.: Prediction-based task migration on S-NUCA many-cores. In: DATE (2019)
20. Sarikaya, R., Buyuktosunoglu, A.: Predicting program behavior based on objective function minimization. In: IISWC (2007)
21. Smola, A.J., Schölkopf, B.: A tutorial on support vector regression. Statistics and Computing **14**(3), 199–222 (2004)
22. SPEC CPU®. https://www.spec.org/cpu2017/index.html (2017)
23. Srinivasan, S., Kumar, R., Kundu, S.: Program phase duration prediction and its application to fine-grain power management. In: IEEE Computer Society Annual Symposium on VLSI. pp. 127–132 (2013)
24. Vashistha, A., Verma, P.: A literature review and taxonomy on workload prediction in cloud data center. In: Confluence (2020)
25. Wang, X.: Pydlm user manual. https://pydlm.github.io/ (2016)
26. Zaman, M., Ahmadi, A., Makris, Y.: Workload characterization and prediction: A pathway to reliable multi-core systems. In: IOLTS (2015)
27. Zhu, Y., et al.: The swiss army knife of time series data mining: Ten useful things you can do with the matrix profile and ten lines of code. Data Mining and Knowledge Discovery **34**(4), 949–979 (2020)
28. Zimmerman, Z., et al.: Matrix profile XIV: Scaling time series motif discovery with GPUs to break a quintillion pairwise comparisons a day and beyond. In: SoCC (2019)