Proceedings of the Federated Conference on Computer Science and Information Systems pp. 7–10 DOI: 10.15439/2020F149 ISSN 2300-5963 ACSIS, Vol. 21

Superiority of Simplicity: A Lightweight Model for Network Device Workload Prediction

Alexander Acker*†, Thorsten Wittkopp*†, Sasho Nedelkoski*, Jasmin Bogatinovski*, Odej Kao*
* Technische Universität Berlin, Germany
{alexander.acker, t.wittkopp, sasho.nedelkoski, jasmin.bogatinovski, odej.kao}@tu-berlin.de
† Alphabetic order, equal contribution

Abstract*—The rapid growth and distribution of IT systems increases their complexity and aggravates operation and maintenance. To sustain control over large sets of hosts and the connecting networks, monitoring solutions are employed and constantly enhanced. They collect diverse key performance indicators (KPIs) (e.g. CPU utilization, allocated memory, etc.) and provide detailed information about the system state. Predicting the future progress of those KPIs allows ahead of time optimizations like anomaly detection or predictive maintenance and can be defined as a time series forecasting problem. Although, a variety of time series forecasting methods exist, forecasting the progress of IT system KPIs is very hard. First, KPI types like CPU utilization or allocated memory are very different and hard to be modelled by the same model. Second, system components are interconnected and constantly changing due to soft- or firmware updates and hardware modernization. Thus a frequent model retraining or fine-tuning must be expected. Therefore, we propose a lightweight solution for KPI series forecasting. It consists of a weighted heterogeneous ensemble method composed of two models - a neural network and a mean predictor. As ensemble method a weighted summation is used, whereby a heuristic is employed to set the weights. The modelling approach is evaluated on the available FedCSIS 2020 challenge dataset and achieves an overall R^2 score of 0.10 on the preliminary 10% test data and 0.15 on the complete test data. We publish our code on the following github repository: *https://github.com/citlab/fed_challenge

I. Introduction

IT systems are rapidly evolving to meet the growing demand for new applications and services in a variety of fields like industry, medicine or autonomous transportation. This entails an increasing number of interconnected devices, large networks and growing data centres to provide the required infrastructure. Although accelerating innovations and business opportunities, this trend increases complexity and thus, aggravates the operation and maintenance of these systems. Operators are in need of assistance to be able to maintain control over this complexity. Therefore, monitoring solutions are implemented. They constantly collect system KPIs like latency, throughput, or system resource utilization and provide detailed information about the monitored IT system. One particularly important aspect of system monitoring is the prediction of future system load. Several efforts where made to enable this ranging from linear regression [1], Bayesian statistics [2] and neural networks [3].

A precise prediction of future system load enables ahead of time decision making. An anomaly detection methods can be employed to compare the difference between the predicted and the actual state and raise alarms in case of unforeseen deviations [4]. Furthermore, scheduling decision [5], network routing and dimensioning [6], data centre cooling control [7] or predictive maintenance [8] all benefit from precise system load predictions.

The task of system load prediction can be formulated as a time series forecasting problem but comes with specific challenges. First, different KPI types are highly non-uniform. CPU utilization is usually very volatile, memory allocation is rarely overlaid by noise and disk read and write operations expose bursty patterns due to buffering resulting in flat sequences with sporadic peaks. The concrete pattern of these series depend of partly unknown external and a variety of internal factors. There are temporal dependencies night- and daytime hours or occasional events like Christmas days influencing the system load. Also, the IT system itself is problematic from modeling perspective due to their dynamic nature and high uncertainty. Frequent soft- and firmware updates or hardware modernization change system properties and usually require model retraining or fine-tuning. This imposes the requirement of frequent and fast model adaption.

Related work on time series forecasting is diverse and ranges from traditional linear or non-linear regression [9], stochastic methods [10], deep learning models [11] and ensemble methods [12], [13]. Traditional regressive or statistical models are often not able to capture the underlying complex processes while neural networks or ensemble methods suffer from high complexity and an accompanying high computational overhead.

Considering this, we present our solution for this years FedCSIS 2020 challenge [14], which is a model for network device workload prediction. It combines the overall average of each KPI series with a prediction from a linear neural network. Furthermore, we employed heuristics to tackle numerical imprecision and enhance overall prediction performance. Our solution achieved an overall R^2 score of 0.105 on the preliminary 10% test data and 0.15 on the complete test data.

The rest of the paper is structured as follows. Section II provides a preliminary analysis of the problem and available training data set. Section III introduces our solution for workload prediction. It includes a formal problem definition and explains each element of our proposed method. An evaluation is performed and results are presented in section IV. Finally section V concludes our paper.

II. Network Device Workload Prediction

This year FedCSIS 2020 challenge [14] was to predict the future workload of network devices based on past workload observations. More specifically, the workload of a set of devices, referred to as hosts, were characterized by KPI series such as CPU utilization, incoming and outgoing network traffic or allocated main memory. The data were collected hourly over a period of 3 months with sporadically missing samples. Overall, 45 different KPIs were recorded from 3,716 hosts, whereby the workload of individual hosts was described by different KPI subsets. Each hourly KPI series sample consists of seven aggregated measurements. These are the number of collected samples, the mean and standard deviation, the first, last, highest and lowest measurement. Out of the seven aggregations only the mean value must be predicted, resulting in a possibly multivariate input but univariate output.

The plots in Fig. 1 show four different KPI mean values from six different hosts. Thereby, the series was split into weekly windows from Monday until Sunday and arranged by the hour of the week resulting in ten aggregated weekly series for each plot. The dark line shows the mean value while the light line visualizes the 0.95 confidence interval. It can be observed that KPI series are highly non-uniform, which indicates the major challenge when faced with forecasting the expected future values of the KPIs.

III. Lightweight Workload Prediction Model

In this section we present our method for lightweight workload prediction. Its concept and architecture were chosen based on the previously described observations and analyses in section II.

A. Preliminaries

We define the task of workload prediction as a time series forecasting problem. A time series is an temporally ordered sequence of values $X = (X_t(\cdot) \in \mathbb{R}^d : t = 1, 2, \ldots, T)$, where d is the dimensionality of each point. For $X_b^a(\cdot) = (X_a(\cdot), X_{a+1}(\cdot), \ldots, X_b(\cdot))$, we denote indices a and b with $a \leq b$ and $0 \leq a, b \leq T$ as time series boundaries in order to slice a given series $X_T^0(\cdot)$ and acquire a subseries $X_b^a(\cdot)$. The variable T defines the time stamp of the last sample of the past observations. Additionally, we use the notion $X(i)$ to refer to a certain dimension i, with $1 \leq i \leq d$. Furthermore, meta information for each time series value $X_t(\cdot)$ are denoted as M_t.

The problem of workload prediction is modelled as the forecasting of a future univariate value $X_{T+w}(i)$, with $w \geq 1$, conditioned on a sequence of past values $X_T^0(\cdot)$, and known meta information about the future time stamp M_{T+w}. Therefore, the learning objective is to select a function $h : \mathbb{R}^N \mapsto \mathbb{R}$, where N is the dimensionality of the input, that results in a small generalization loss:

$$\mathcal{L} = \frac{1}{|\mathcal{W}|} \sum_{w \in \mathcal{W}} L(h(X_T^0(\cdot), M_{T+w}), X_{T+w}(\cdot)). \quad (1)$$

Thereby, L is a bounded loss function and $\mathcal{W}$ is the set of offsets defining all future time stamps to predict.

B. Lightweight Workload Prediction Model

The overall architecture of our method is depicted in Fig. 2. A future time series value $X_{T+w}(i)$ should be predicted based on the history $X_T^0(\cdot)$ and its known meta information M_{T+w}. For the task of workload prediction, each time series X represents an KPI. The respective dimensions of samples $X_t(\cdot)$ are aggregated values of that KPI between time $t-1$ and t. Due to their importance, we selectively define the mean and last measurement as $\overline{x}_t$ and $x_t^{(l)}$, where $\overline{x}_t, x_t^{(l)} \in X_t(\cdot)$. The mean value of the sample $\overline{x}_{T+w} \in X_{T+w}(\cdot)$ is the prediction target. Since many workload series are seasonal, we additionally add the encoded day of week and hour of day as meta information M_{T+w}. Subsequently, each model element is described in detail.

Preprocessing. Initially, a rescaling of each value in the KPI series $X_T^0(\cdot)$ to a fixed upper bound d and a respectively linear scale to the lower bound is performed. Furthermore, values in $X_T^0(\cdot)$ are expected to be sampled hourly. If samples are missing, a linear interpolation is employed.

Feature Selection. Due to the additional overhead that is introduced by automated feature selection methods, we choose to select a fixed subset of features manually. The features are selected depending on the model that they are forwarded to. Therefore, we define a filter F_1 for the mean predictor and a filter F_2 for the neural network model (NN). The filter F_1 includes only the mean values of $X_T^0(\cdot)$. Filter F_2 applies two feature selection operations. First, out of the aggregated values in the last available series sample, we pick the mean and last value, i.e. $\overline{x}_T, x_T^{(l)} \in X_T(\cdot)$. Second, motivated by the seasonality of system load, we additionally use the mean value of the same hour of the week as the prediction target of previous k weeks.

The Models. The mean predictor calculates the overall average over the filtered sample series $F_1(X_T^0(\cdot))$. As NN a linear feed-forward neural network is used. It receives the preprocessed and filtered data $F_2(X_T^0(\cdot))$, the meta-information values M_{T+w} and the output of the mean model. These are combined to a flat input vector $\mathbf{x}$. The learning objective is to minimize the squared error loss between the prediction and the mean value of $\overline{x}_{T+w} \in X_{T+w}(\cdot)$:

$$L = (h(\mathbf{x}) - \overline{x}_{T+w})^2. \quad (2)$$

The proposed NN architecture is a fanning out first hidden layer. The subsequent layers are tampered, which works as regularization. We use a dropout between the first and second hidden layer as an additional regularization. A rectifier linear unit (ReLU) activation is applied to the output value of the network. The output of the mean model and NN model are respectively denoted as $o_{T+w}^{(1)}$ and $o_{T+w}^{(2)}$.

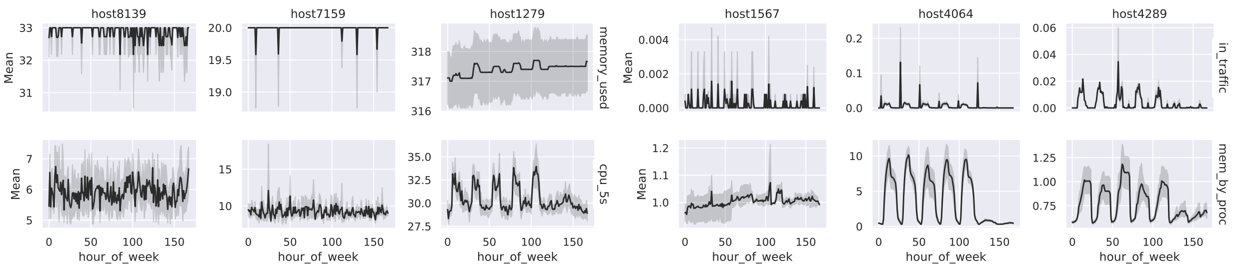

Fig. 1. Example of four KPIs for six hosts. A great in-between and within KPI value diversity for the different hosts can be observed.

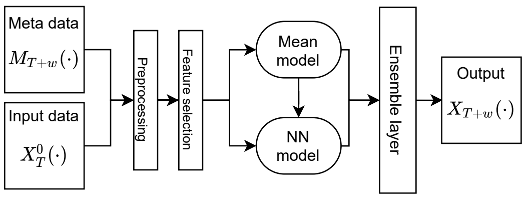

Fig. 2. Overall solution architecture.

Ensemble Layer. To combine the predictions of the mean model and the NN model, a weighted average over the model outputs is calculated:

$$o_{T+w} = \sum_{o^{(i)}_{T+w} \in \{o^{(1)}_{T+w}, o^{(2)}_{T+w}\}} w_i o^{(i)}_{T+w}, \text{where} \sum_i w_i = 1. \quad (3)$$

The usage of two models is motivated by the non-uniformity of KPI series. While the neural network is capable to predict seasonal series fairly well, it fails to accurately predict constant but noisy series. A simple average over all mean metrics of a KPI resulted in good predictions for constant but noisy series but resulted in bad predictions for seasonal series. By combining both, we expect to achieve a generally better result.

IV. Evaluation

Based on the provided dataset, the future progress of 10,000 KPI series must be predicted. Samples are sampled hourly. This results in a sequence of 168 samples that have to be predicted for each series. In this section, we evaluate the proposed method in terms of runtime and prediction performance.

A. Training and Parameterization

KPI series are diverse depending on the type and the host from which they were collected. Therefore, we choose to train individual models for each KPI series. The mean predictor calculates an overall mean over all mean values from the available three months of data.

Training of the NN requires the definition of a training set. Therefore, a set of inputs and prediction targets are defined. The target is always a specific mean value $\overline{x}_{t_p} \in X_{t_p}(\cdot)$ at prediction target time stamp $t_p \leq T$. The hour of day $m_1 \in \{1, 2, \dots, 24\}$ and day of week $m_2 \in \{1, 2, \dots, 7\}$ are defined as meta information $M_{t_p} = \{m_1, m_2\}$. This KPI training series slice is defined as $X_e^s(\cdot)$ with $e = t_p - ((m_2 - 1) * 24 + m_1)$ and $s = e - 168 * k$, where 168 are the hours of one week, $s \geq 0$ and $k \geq 1$. Thereof, the mean and last value from the last sample are selected $\overline{x}_e, x_e^l \in X_e(\cdot)$. Further, respecting the seasonality of several KPI series, the mean value of the same hour of the week as the prediction target is added to the input. These can be accessed via $\{\overline{x}_\tau \in X_\tau(\cdot) : \tau = t_p - i * 168, i = 1, 2, \dots, k\}$

To create the training data we set $k = 2$. For the rescaling, we define $d = 100$. Training of the NN is done via backpropagatuion on the mean square error as optimization criterion and Adam as the optimizer. We set the learning rate to $10e^-3$ and use dropout probability of 0.1.

B. Runtime Analysis

A preliminary runtime analysis is conducted where our neural network is compared to a recurrent version of it. For the recurrent network, we use long short term memory (LSTM) instead of linear cells. We measure the training time per epoch on a bare-metal machine with an Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz, 3x32 GB RAM and two Nvidia GeForce RTX 2080 Titan GPUs whereof one was utilized during the runtime measurement experiments. Ubuntu 18.04.3 LTS with kernel version 5.3.0-51-generic is installed as OS and Python version 3.6.7 and PyTorch version 1.4.0 are used to implement the networks.

The LSTM version requires significantly more time for training than the network with linear cells. In comparison, the runtime increases by a factor of ten. The mean training runtime per epoch of the linear version is 2.37 seconds per epoch with a standard deviation of 0.03 and 0.95 confidence interval of $[2.38, 2.37]$. For the network version with LSTM cells a training time per epoch of 25.72 seconds per epoch is measured with a standard deviation of 0.18 and 0.95 confidence interval of $[25.74, 2.70]$. Having six training epochs per series and a total number of $10,000$ series means a total required training time of 39.5 hours for the linear cells and 17.9 days when using LSTM cells.

Although recurrent neural network architectures especially with LSTM cells are reported to perform well on sequential data prediction tasks [15], our runtime analysis shows that the required training time is very high and considered as infeasible.

TABLE I
R^2 SCORES OF BEST THREE SUBMISSIONS TOGETHER WITH THE BASELINE.

	baseline	1st	2nd	Ours
Preliminary test set (10%)	0.2267	0.1888	0.1841	0.1053
Complete test set (100%)	0.2295	0.163	0.1515	0.1501

C. Prediction Results

The performance of the proposed workload prediction method is evaluated against the withheld test set by submitting the solution via the official FedCSIS 2020 challenge submission system. The submissions are scored by the R^2 score defined as

$$R^2 = 1 - \frac{\sum_t (\overline{x}_t - o_t)^2}{\sum_t (\overline{x}_t - \overline{\overline{x}})^2}, \tag{4}$$

where $\overline{x}_t \in X_t(\cdot)$ and $\overline{\overline{x}}$ as the overall average over all mean samples. Based on our observation several KPI series are mainly constant with sporadic deviations, resulting in a very small normalization value (denominator of in Eq. 4). This results in high division values and thus, low R^2 scores even for small deviations of the predicted values. These values had a negative impact on the overall R^2 score. Furthermore, several KPI series can be described as the noise around a baseline. This motivates us to implement a heuristic to choose an adaptive weighting of the model outputs. First, the neural network is trained. Second, the last available week is used as a prediction target and the data before that week as input. Since this last week was explicitly trained on, we assume precise prediction results, i.e. R^2 score close to 1. If the neural network output resulted in a lower score than the output of the average predictor, we set the weight for the average predictor to 1.0 and the neural network weight to 0.0. Otherwise, the both weights were set to 0.5.

Finally, the prediction of the submission is done based on the k last available weeks in the training data set. The R^2 scores of the best three submissions are listed in TABLE I. None of the submitted results is able to achieve the specified baseline. Two submissions achieved a better R^2 score than our solution with 0.1888 and 0.1841 on the preliminary 10% of test data and 0.163 and 0.1515 on the complete test dataset. With our proposed lightweight model, we achieve an R^2 score of 0.1053 on the preliminary 10% test data and 0.1501 on the complete test dataset. We did not carry out any attempts to optimize for the 10% preliminary test data since it was not clear whether it is a general representation of the complete test dataset. Therefore, it is interesting to observe that our solution is the only one achieving a better score on the complete dataset than on the preliminary 10%.

V. Conclusion

We tackle the given challenge of network device workload prediction based on KPI data with a lightweight model that ensembles the predictions of a neural network and a mean predictor. The ensemble is done by a weighted summation. A heuristic is used to selectively set the weights for each model prediction. The lightweight nature of the method allows training individual models for each KPI series respecting the diverse nature of different KPI types and host. Furthermore, frequent retraining is feasible with the proposed solution.

We provide a runtime analysis between LSTM cells and linear cells showing revealing the usage of LSTM cells as infeasible. We evaluate our solution against the FedCSIS 2020 challenge dataset. The experiment results show that the lightweight approach predicts future KPI values with an overall R^2 score of 0.105 on the preliminary 10% test data and 0.15 on the complete test data.

For future work, we see further experimentation with different network types like convolutional neural networks or attention mechanisms as promising. Furthermore, the learning of the summation weights when aggregating mean predictor and neural network outputs are sources for potential optimization.

References

[1] P. A. Dinda and D. R. O'Hallaron, "Host load prediction using linear models," *Cluster Computing*, vol. 3, no. 4, pp. 265–280, 2000.

[2] S. Di, D. Kondo, and W. Cirne, "Host load prediction in a google compute cloud with a bayesian model," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[3] B. Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, "Host load prediction with long short-term memory in cloud computing," *The Journal of Supercomputing*, vol. 74, no. 12, pp. 6554–6568, 2018.

[4] F. Schmidt, F. Suri-Payer, A. Gulenko, M. Wallschläger, A. Acker, and O. Kao, "Unsupervised anomaly event detection for vnf service monitoring using multivariate online arima," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 278–283.

[5] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 2876–2880.

[6] A. Howard, A. Zhmoginov, L.-C. Chen, M. Sandler, and M. Zhu, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," 2018.

[7] F. Ahmad and T. Vijaykumar, "Joint optimization of idle and cooling power in data centers while maintaining response time," *ACM Sigplan Notices*, vol. 45, no. 3, pp. 243–256, 2010.

[8] M. Yaseen, D. Swathi, and T. A. Kumar, "Iot based condition monitoring of generators and predictive maintenance," in *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2017, pp. 725–729.

[9] J. H. Stock and M. W. Watson, "A comparison of linear and nonlinear univariate models for forecasting macroeconomic time series," National Bureau of Economic Research, Tech. Rep., 1998.

[10] M. R. Hassan and B. Nath, "Stock market forecasting using hidden markov model: a new approach," in *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*. IEEE, 2005, pp. 192–196.

[11] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *arXiv preprint arXiv:1912.09363*, 2019.

[12] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[13] X. Qiu, Y. Ren, P. N. Suganthan, and G. A. Amaratunga, "Empirical mode decomposition based ensemble deep learning for load demand time series forecasting," *Applied Soft Computing*, vol. 54, pp. 246–255, 2017.

[14] A. Janusz, M. Przyborowski, P. Biczyk, and D. Slezak, "Network Device Workload Prediction: A Data Mining Challenge at Knowledge Pit," in *Proceedings of FedCSIS 2020, Sofia, Bulgaria*, 2020.

[15] S. Nedelkoski, J. S. Cardoso, and O. Kao, "Anomaly detection and classification using distributed tracing and deep learning." in *CCGRID*, 2019, pp. 241–250.