Estimating Application Performance in Container-Based Environments: A Cross-Domain Monitoring Approach

Marcus V. Diniz dos Reis[1], Rodrigo Moreira[1](✉), Larissa F. Rodrigues Moreira[1], and Flávio de Oliveira Silva[2]

[1] Federal University of Viçosa (UFV), Minas Gerais, Brazil
{marcus.v.reis,rodrigo,larissa.f.rodrigues}@ufv.br
[2] University of Minho (UMinho), Braga, Portugal
flavio@di.uminho.pt

Abstract. Estimating the application performance in distributed scenarios is challenging due to numerous variables and influences. Combining Artificial Intelligence (AI) with advanced monitoring methods offers a promising approach for predicting application quality. State-of-the-art solutions leverage robust monitoring and AI algorithms, such as Deep Neural Networks (DNNs), to detect temporal relationships and improve estimation accuracy. However, these methods often require significant computational resources and impose overheads due to the fine-grained monitoring requirements. This paper proposes an alternative hypothesis: effective insight into the application's behavior and user perception does not rely solely on highly granular metrics. Instead, lightweight AI algorithms combined with generic infrastructure metrics–such as computing, networking, storage, and operating system variables–can yield valuable patterns. Our method achieved lower Mean Absolute Error (MAE) and Mean Squared Error (MSE) in forecasting write-and-read operation latencies in a database cluster, demonstrating its effectiveness and efficiency.

1 Introduction

Modern applications are seamlessly deployed on complex infrastructure to achieve high availability, performance, and user satisfaction [1]. To evolve these computing environments and the underlying network infrastructure to satisfy Service-Level Agreements (SLAs), many paradigms have been proposed, such as edge computing, containerization, and network slicing [2,3]. In these computing environments, there are different challenges, such as long queuing time, resource competition, resource idle, and complex management and monitoring [4] while Service-Level Agreement (SLA) satisfaction is a business differentiator.

To cope with SLA requirements, Artificial Intelligence (AI) techniques have been employed in different paradigms to support the management and orchestration of applications in high-performance computing infrastructures and have been effective in addressing many challenges [5,6]. Furthermore, production environments are dynamic, requiring an efficient, low-overhead monitoring method capable of correlating application performance with computing resources using generic infrastructure metrics [7].

In the literature, some approaches employ sophisticated Machine Learning (ML) algorithms, such as those based on Deep Neural Networks (DNNs), to forecast the performance of network-based applications despite their high computational requirements [8]. The trade-off between high granularity and good performance versus low granularity and low resource consumption. Consequently, there is an opportunity to investigate low-granularity methods that offer satisfactory performance without compromising the ability to estimate the impact of external variables on application performance.

This paper presents an approach based on cross-domain metrics union, from computing, network, and operating system metrics, to estimate application performance in container-based environments. Through a distributed architecture that identifies patterns in resource usage and relates them to application behavior, we provide (1) a non-intrusive method to estimate performance using generic infrastructure monitoring data and (2) validate the model in real-world scenarios, showing its effectiveness in predicting resource usage.

The remainder of this paper is organized as follows: Sect. 2 presents the approaches correlated with the proposal, while Sect. 3 presents the aggregation method of metrics between domains. Section 4 presents our proposal's main findings, insights, and future work and conclusions in Sect. 5.

2 Related Work

Many approaches in the literature can estimate the performance of applications using AI in different domains [9], such as cloud-native or container-based approaches. Effective forecasting depends on the data's quality, detail, and temporal relationship. Achieving this requires sophisticated monitoring approaches that can add more overhead to the cloud infrastructure [10]. Our contribution lies in using generic monitoring metrics to estimate the application performance in container-based environments accurately.

Tam et al. [11] proposed the PERT-GNN, a transformer based on Graph Neural Networks (GNN) and inspired by a statistical tool for project management called Program Evaluation and Review Technique (PERT). PERT-GNN is a regression-based model for accurately predicting latency in containerized applications. It was evaluated using datasets from DeathStarBench and Alibaba production clusters. This method involves creating PERT graphs from microservice traces, training the PERT-GNN model, and benchmarking it against existing methods. The results show that PERT-GNN outperforms the other approaches, with a Mean Absolute Percentage Error (MAPE) below 12%, demonstrating its latency prediction effectiveness.

Wang et al. [12] introduced a load prediction-driven scheduling strategy for container cloud environments using a CNN-BiGRU-Attention model for workload forecasting. The model predicts continuous load values using the cluster-trace-v2018 dataset, which includes data from 4000 machines over eight days. Evaluation metrics such as Mean Absolute Error (MAE), MAPE, and Root Mean Square Error (RMSE) show that the model significantly enhances prediction accuracy and efficiency in container cloud environments.

Al Qassem et al. [13] proposed a proactive autoscaling method using a Random Forest model to predict CPU and memory utilization. The model forecasts continuous resource usage values with the fastStorage dataset, containing traces from 1,250 Virtual Machines (VMs). Evaluation using metrics such as MAE, RMSE, and R^2 demonstrates that the model improves resource allocation efficiency and reduces latency in containerized microservice environments.

Wang et al. [14] proposed a latency prediction method using GNN and Profile Engine, focusing on resource utilization estimation. The dataset includes DeathStarBench and HPC-AI500. Evaluation using MAPE shows that the system significantly improves resource estimation accuracy and efficiency in microservice environments.

Taha et al. [15] presented a proactive auto-scaling system using a hybrid MLP-LSTM model to predict resource demands for Service Function Chains (SFCs) in cloud environments. The system forecasts CPU, memory, and bandwidth utilization for Virtual Network Functions (VNFs) using the Clearwater IMS and Web SFC datasets. Evaluated with MAE, MSE, and RMSE metrics, the MLP-LSTM model outperforms others in prediction accuracy and adaptability, reducing under-provisioning and over-provisioning in dynamic cloud environments.

Mondal et al. [16] proposed a proactive scaling scheme for Kubernetes using a Gated Recurrent Unit (GRU) model to predict system load and optimize resource allocation. The model forecasts future CPU usage using the Google-cluster-data-2011-2 dataset. The evaluation shows that the GRU-based autoscal Table 1 outlines the approaches in container-based environments and resource efficiency.

Silva et al. [17] presented an online machine learning-based auto-scaling subsystem for edge computing, combining reactive and proactive strategies via the MAPE-K loop. The method predicts workload for elastic resource allocation using regression, with datasets including the Madrid Traffic Dataset and synthetic data from TimeSynth. Experiments in a hierarchical edge environment for video analytics show reduced SLA violations, minimized waste, and improved provisioning accuracy compared to baseline methods.

Table 1 outlines the approaches in container-based environments. Each column provides specific insights: Method describes the prediction techniques used; Task indicates whether it's regression or classification; Dataset identifies the data sets employed; Evaluation Metrics highlight performance measures like MAE, Mean Squared Error (MSE), and MAPE; Data Source reveals the origin of data, such as real-time monitoring or ElasticSearch; and Data Granularity offers a qualitative assessment of data detail richness, categorized as high, medium, or low. Data granularity is a qualitative metric that indicates the level of detail of a dataset. A high granularity reflects fine-grained data from sophisticated platforms and data pipelining. Medium granularity balances detail, whereas low granularity uses generalized data, offering broader overviews but eventually less precision.

Unlike the studies mentioned above, which depend on fine-grained monitoring and resource-intensive AI models [11,14,16], our approach focuses on efficiency using generic infrastructure metric aggregation. Although the existing methods achieve high accuracy, they may incur significant overhead. By contrast, our approach achieves accurate latency predictions with reduced data and computational demands using basic

Table 1. Short state-of-the-art comparison.

Approach	Method	Dataset	Evaluation Metrics	Data Source	Data Granularity
Tam et al. [11]	GNN using PERT to predict end-to-end latency in microservices.	Benchmarks, and large-scale Alibaba traces.	MAE, MAPE	Distributed monitoring tools such as Jaeger, and Prometheus.	High
Wang et al. [12]	CNN-BiGRU-Attention model for load prediction.	cluster-trace-v2018	MAE, MAPE, RMSE	Real-time monitoring.	High
Al Qassem et al. [13]	Random Forest model for predicting CPU and memory utilization.	fastStorage dataset	MAE, RMSE, R	Real-time monitoring.	High
Wang et al. [14]	Latency prediction using GNN and RNN-based resource estimation system.	DeathStarBench, and HPC-AI500.	MAPE	Real-time monitoring with Prometheus and Jaeger.	Low
Taha et al. [15]	Latency and resource demand prediction using hybrid MLP-LSTM with online learning and error correction.	Clearwater IMS, and Web SFC.	MAE, MSE, RMSE	Monitoring data via Grafana.	Medium
Mondal et al. [16]	GRU model for latency prediction.	Google-cluster-data-2011-2	MSE, RMSE, MAE, R	Monitoring data from Google Cluster.	Medium
Silva et al. [17]	Online ML for latency prediction and container adjustment based on workload changes.	Real and synthetic data from a video analytics application.	R, MAE, MSE, QoS degradation, SLA violations, Elasticity Speedup.	Real-time monitoring.	High
Ding et al. [18]	MECE model for latency prediction using resource and management cost data.	JPetStore-6, and SpringBlog.	MAE, RMSE	Real monitoring data using Prometheus and JMX Exporter.	Medium
Ours	**Generic Monitoring Metrics feeding basic ML algorithms.**	**Generated by ourselves**	**MAE, MSE, and MAPE**	**NetData**	**Low**

ML algorithms, offering a more resource-efficient solution for container-based environments.

3 Proposed Method

Our method consists of joining generic metrics from a monitoring platform (NetData) to fit and generalize application latency forecasting. Figure 1 presents a diagram of the method used in this study and details the main steps of the developed process.

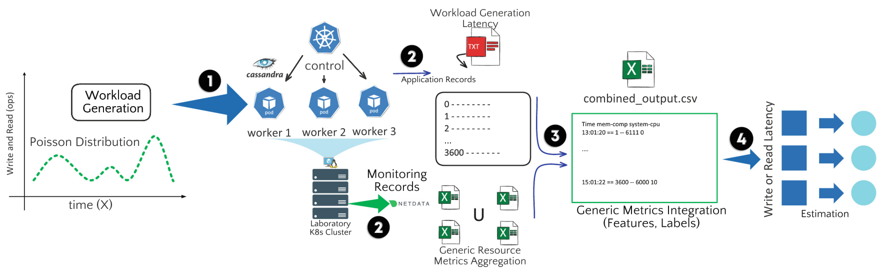


Fig. 1. Proposed Aggregation and Evaluation Method.

❶ **First Phase**: we generated a workload (with `cassandra-stress`) for 60 min. This load was generated separately for Write and Read operations, with an average arrival rate of $\lambda = 25$ clients per minute, modeled by a Poisson distribution defined by the probability function in Eq. 1. We modeled this synthetic workload generation with sinusoidal behavior to approximate real scenarios where there are workloads with seasonal behavior.

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{1}$$

where k represents the number of events and instantiations of clients constantly performing Write or Read operations at a fixed interval. Additionally, the arrival rate varied sinusoidally over time, with amplitudes $A = 20$ and $P = 10$ min, as described by Eq. 2.

$$\lambda(t) = \lambda_0 + A \cdot \sin\left(\frac{2\pi}{P}t\right) \tag{2}$$

Here, $\lambda_0 = 25$ represents the average arrival level, and t is the time in min. The execution logs were stored in a file with a timestamp to link with generic monitoring metrics. The operations were distributed across multiple Cassandra instances deployed as pods in the Kubernetes environment. The pods were responsible for handling read and write tasks simultaneously. This approach used Kubernetes scalability and resource management to orchestrate operations efficiently. Each pod operated in isolation but collaboratively, allowing a detailed collection of read and write latencies in a distributed manner. This architecture made monitoring the load's impact on the system performance easier, ensuring greater flexibility and adaptability in execution and data collection.

❷ **Second Phase:** Using the NetData system, read and write data were collected separately to obtain detailed metrics (Comma-Separated Values (CSV)) of computational resource usage during specific execution moments. This process enabled the creation of a representative dataset that correlates system performance with computational resource consumption at different execution times. Our cross-domain metric-joining criteria are based on timestamps, where each monitored domain—such as compute, network, or operating system–has a timestamp for its records. We then performed an inner join to build the feature set.

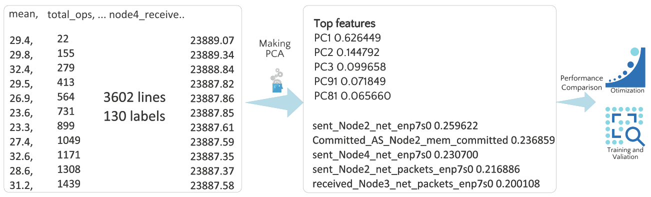
❸ **Third Phase:** for data aggregation and analysis, the previously generated feature set, which initially contained more than 130 labels (resulting from metric-joining), was further enriched by incorporating application latency timestamps collected during the workload generation phase. We then applied Principal Component Analysis (PCA) to reduce the dataset's dimensionality and optimize the analysis. The evaluation of the principal components determined that the most relevant features were associated with the variables `net_enp7s0` and `netpackets`. Based on these results, the dataset was reduced to 16 features, preserving the most significant information for the study's objectives.

❹ **Fourth Phase:** using the reduced dataset, we evaluated four lightweight machine learning algorithms: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Random Forest (RF), Decision Tree (DT) and Long Short-Term Memory

(LSTM). The Grid Search technique was applied to each model to optimize the hyperparameters and obtain the best performance in previously defined metrics.

4 Evaluation and Discussion

In an hour-sized generated dataset, we performed a PCA analysis, as shown in Fig. 2, using the most relevant features and optimized parameters, we proceed with training of the basic ML algorithms (such as RF, KNN, DT, and SVM) to assess their ability to predict the application latency in container-based environments.



```
mean,   total_ops, ... node4_receive..

29.4,     22                        23889.07
29.8,     155                       23889.34
32.4,     279                       23888.84
29.5,     413                       23887.82
26.9,     564      3602 lines       23887.86
23.6,     731      130 labels       23887.85
23.3,     899                       23887.61
27.4,     1049                      23887.59
32.6,     1171                      23887.35
28.6,     1308                      23887.37
31.2,     1439                      23887.58
```

```
Top features
PC1 0.626449
PC2 0.144792
PC3 0.099658
PC91 0.071849
PC81 0.065660

sent_Node2_net_enp7s0 0.259622
Committed_AS_Node2_mem_committed 0.236859
sent_Node4_net_enp7s0 0.230700
sent_Node2_net_packets_enp7s0 0.216886
received_Node3_net_packets_enp7s0 0.200108
```

Fig. 2. Our evaluation workflow.

Table 2 highlights the hyperparameters tested for each model.

Table 2. Tested hyperparameters for each model – best ones highlighted.

Model	Hyperparameter Search Space					
RF	*No. Estimators*	*Max Depth*	*Min Samples Split*	*Min Samples Leaf*	*Max Features*	*Bootstrap*
	[50, 100, **200**]	[None, **10**, 20, 30]	[2, 5, **10**]	[1, 2, **4**]	[auto, **sqrt**, log2]	[**True**, False]
KNN	*No. Neighbors*		*Weights*		*Metric*	
	[3, 5, 7, **10**]		[**uniform**, distance]		[euclidian, **manhattan**]	
SVM	*l*		*Kernel*		*Gamma*	
	[**0.1**, 1, 10]		[linear, **poly**, rbf, sigmoid]		[scale, **auto**]	
DT	*Max Depth*		*Min Samples Split*		*Min Samples Leaf*	
	[None, **10**, 20, 30]		[2, 5, **10**]		[**1**, 2, 4]	

We empirically defined the parameters for LSTM considering training with 50 epochs, the ReLU activation function, Adam optimizer with a learning rate of 0.001, and a loss function set to MSE. LSTM was evaluated merely as a baseline for our cross-domain metric aggregation. Our evaluation focuses on lightweight ML algorithms applied to a generic monitoring dataset of container-based infrastructure.

The performance of the forecasting algorithms was evaluated using the MAE, MSE, MAPE, and the Coefficient of Determination (R^2) metrics. The MAE measures the average magnitude of the errors and is given by:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|,$$

where y_i and $\hat{y}_i$ are the observed and predicted values, respectively, and n is the total number of observations. The MSE penalizes larger errors and is expressed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

The MAPE evaluates the relative prediction accuracy as a percentage:

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|.$$

Finally, the R^2 metric assesses the proportion of variance explained by the model:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where $\bar{y}$ is the mean of the observed values. These metrics provide comprehensive insights into the accuracy and reliability of the forecasting models.

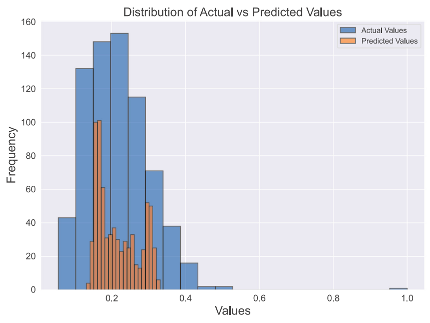
Subsequently, we conducted training and testing on the data in proportion (80–20%). Table 3 compares the different ML models for Write and Read operations. The metrics analyzed include MSE, the coefficient of determination (R^2 Score), MAE, and MAPE. These metrics provide a comprehensive view of model accuracy, error distribution, and predictive reliability.

Table 3. Model Comparison Based on Different Metrics (Write and Read).

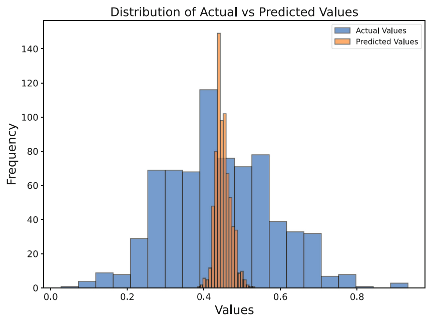
Metric	RF	KNN	SVM	DT	LSTM
Write MSE	0.0042	0.0045	0.0044	0.0051	0.0046
Read MSE	0.0198	0.0216	0.0193	0.0241	0.0196
Write R^2 Score	0.4038	0.3642	0.3749	0.2840	0.3442
Read R^2 Score	−0.0258	−0.1214	−0.0001	−0.2518	−0.0172
Write MAE	0.0466	0.0477	0.0478	0.0517	0.0492
Read MAE	0.1123	0.1164	0.1111	0.1227	0.1125
Write MAPE	25.60%	25.21%	26.98%	27.59%	25.50%
Read MAPE	33.97%	34.74%	33.48%	36.33%	34.83%

In our experimental scenario, RF performed better in forecasting write data, presenting the lowest MSE (0.0042) and MAE (0.0466), in addition to a high R^2 (0.4038). These data indicate that the RF model captured the relationships in the write data, providing consistent predictions while requiring low computational resources and training time. For Read operations, performance was more challenging for all models, but SVM had a lower MSE (0.0193) and MAE closer to ideal (0.1111), although R^2 was still close to zero.

The data distribution graph illustrated in Fig. 3(a) and 3(b) demonstrates how the data are distributed among the different features of our dataset, highlighting load patterns and efficiency in resource management. This indicates that there may be a pattern related to the amount of resources requested using network resources.



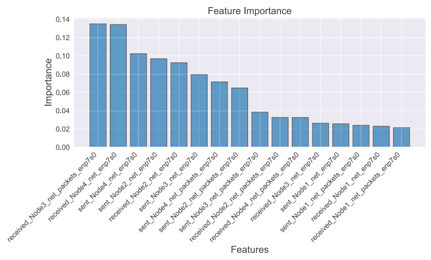
(a) Write



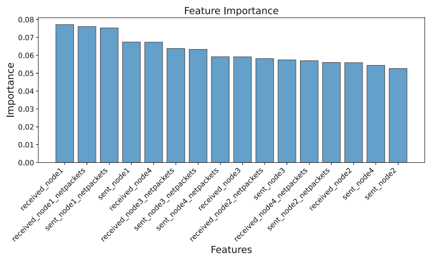
(b) Read

Fig. 3. Analysis of the quality of the dataset generated in our experimental scenario.

Figure 4(a) and 4(b) show the most important features of the solution. The graph indicates that even the most important features still have low values, but the models use a combination of information between the features to recognize the values.



(a) Write



(b) Read

Fig. 4. Feature importance.

Figure 5(a) and 5(b) illustrate the parity curves for the written data, demonstrating the relationship between the actual and predicted values. This graph represents the ideal parity line, and the points that align with it indicate perfectly accurate predictions. The difference between the Write and Read data is notable because the write data are more reliable.

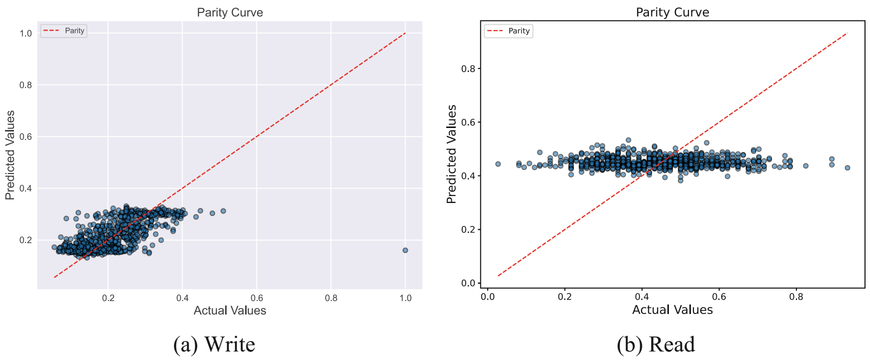


Fig. 5. Parity Curve.

In Fig. 6(a) and 6(b), we can observe the presence of patterns observed by the model and compare them with their real values. The writing data demonstrated a pattern that the model could understand and follow. The Reading model demonstrated values with a low relationship compared to the reading data.

The Residual Histograms, as shown in Fig. 7(a) and 7(b), show the number of residuals in specific sections. This allowed us to identify values with the most significant disparity in the dataset. As observed in the previous graphs, the writing data demonstrated fewer residuals than the reading data, which showed greater consistency in writing data for regression analyses.

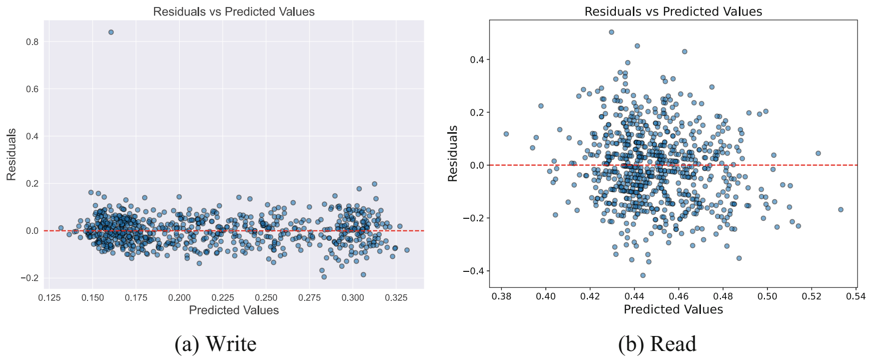


Fig. 6. Residual Graph.

Figure 8(a) and 8(b) show the relationship between the actual (actual) and Predicted Values (predicted values) for the writing and reading processes, respectively. Both graphs show that the model is functional, but there is room for improvement in reducing dispersion, especially when reading data.

The results demonstrate the model's effectiveness in predicting and analyzing data, with write values demonstrating superior performance in regression models compared

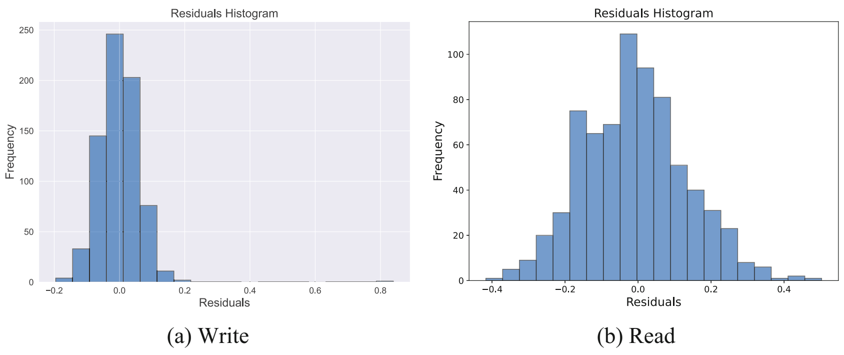


Fig. 7. Residual Histogram.

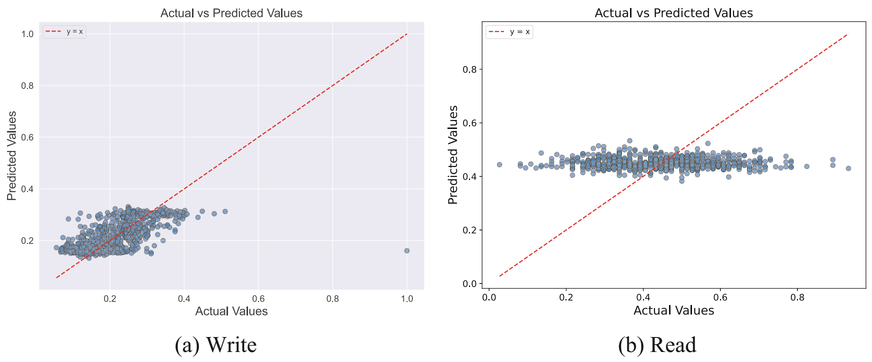


Fig. 8. Scatter Plot.

with read values. Despite the good overall performance, aspects that can be improved to increase prediction accuracy have been identified, particularly in read data. Thus, this study validates a cross-domain metric aggregation approach for inferring application performance in container-based scenarios.

5 Concluding Remarks

This study explored and evaluated the suitability of cross-domain metric aggregation to estimate network application performance in production-ready networks. Using data on network usage and resource allocation, the proposed method demonstrated its ability to provide accurate estimates of computational resource consumption. State-of-the-art solutions are moving towards feeding complex and computation-intensive ML algorithms with data from highly granular monitoring architectures without considering the potential to extract patterns from generic infrastructure metrics.

Alternatively, our method indicates that although it is a computationally feasible approach, employing basic ML algorithms can yield interesting results. Furthermore, it can enrich the management and orchestration platforms for complex infrastructure.

Unlike traditional methods focusing exclusively on specific application domains and algorithms, such as DNNs, the methodology based on multi-domain metric aggregation with basic ML algorithms ensures adaptability and consistency in different standard application environments. Future work will explore the framework and combine other metrics, explore monitoring mechanisms with shorter time intervals, and evaluate their impact on evaluation performance.

Acknowledgments. The authors thank the National Council for Scientific and Technological Development (CNPq) under grant number 421944/2021-8 (call CNPq/MCTI/FNDCT 18/2021), FAPEMIG (Grant #APQ00923-24), FAPESP MCTIC/CGI Research project 2018/23097-3 - SFI2 - Slicing Future Internet Infrastructures, and Fundação para a Ciência e Tecnologia (FCT) within the RD Units Project of Centro ALGORITMI for partially supporting this work.

References

1. Kang, P., Lama, P.: Robust resource scaling of containerized microservices with probabilistic machine learning. In: 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), pp. 122–131 (2020)
2. Moreira, R., Martins, J.S.B., Carvalho, T.C.M.B., Silva, F.d.O.: On enhancing network slicing life-cycle through an AI-native orchestration architecture. In: Advanced Information Networking and Applications (L. Barolli, ed.), (Cham), pp. 124–136, Springer International Publishing (2023)
3. Moreira, L.F.R., et al.: Enabling intelligence on edge through an artificial intelligence as a service architecture. In: 2024 IEEE 13th International Conference on Cloud Networking (CloudNet), pp. 1–8 (2024)
4. Feng, B., Ding, Z.: Application-oriented cloud workload prediction: a survey and new perspectives. Tsinghua Sci. Technol. **30**(1), 34–54 (2025)
5. Carrión, C.: Kubernetes scheduling: taxonomy, ongoing issues and challenges. ACM Comput. Surv. **55** (2022)
6. Shahraki, A., Ohlenforst, T., Kreyß, F.: When machine learning meets Network Management and Orchestration in Edge-based networking paradigms. J. Netw. Comput. Appl. **212**, 103558 (2023)
7. Usman, M., Ferlin, S., Brunstrom, A., Taheri, J.: A survey on observability of distributed edge & container-based microservices. IEEE Access **10**, 86904–86919 (2022)
8. Li, B., et al.: Random sketch learning for deep neural networks in edge computing. Nature Comput. Sci. **1**, 221–228 (2021)
9. Kontopoulou, V.I., Panagopoulos, A.D., Kakkos, I., Matsopoulos, G.K.: A review of ARIMA vs. machine learning approaches for time series forecasting in data driven networks. Future Internet **15**(8) (2023)
10. Wang, T., Xu, J., Zhang, W., Gu, Z., Zhong, H.: Self-adaptive cloud monitoring with online anomaly detection. Futur. Gener. Comput. Syst. **80**, 89–101 (2018)
11. Tam, D.S.H., Liu, Y., Xu, H., Xie, S., Lau, W.C.: PERT-GNN: latency prediction for microservice-based cloud-native applications via graph neural networks. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD ’23, (New York, NY, USA), pp. 2155–2165, Association for Computing Machinery (2023)
12. Wang, L., et al.: An efficient load prediction-driven scheduling strategy model in container cloud. Int. J. Intell. Syst. **2023**(1), 5959223 (2023)
13. Al Qassem, L.M., Stouraitis, T., Damiani, E., Elfadel, I.A.M.: Proactive random-forest autoscaler for microservice resource allocation. IEEE Access **11**, 2570–2585 (2023)

14. Wang, J., Wang, G., Wo, T., Wang, X., Yang, R.: RESCAPE: a resource estimation system for microservices with graph neural network and profile engine. In: 2024 IEEE International Conference on Joint Cloud Computing (JCC), pp. 37–44 (2024)
15. Taha, M.B., Sanjalawe, Y., Al-Daraiseh, A., Fraihat, S., Al-E'mari, S.R.: Proactive autoscaling for service function chains in cloud computing based on deep learning. IEEE Access **12**, 38575–38593 (2024)
16. Mondal, S.K., et al.: Toward optimal load prediction and customizable autoscaling scheme for kubernetes. Mathematics **11**(12) (2023)
17. da Silva, T.P., Neto, A.R., Batista, T.V., Delicato, F.C., Pires, P.F., Lopes, F.: Online machine learning for auto-scaling in the edge computing. Pervasive Mob. Comput. **87**, 101722 (2022)
18. Ding, Z., Xu, Y., Feng, B., Jiang, C.: Microservice extraction based on a comprehensive evaluation of logical independence and performance. IEEE Trans. Software Eng. **50**(5), 1244–1263 (2024)