Modeling Digital Twins of Kubernetes-Based Applications

Davide Borsatti*, Walter Cerroni*, Luca Foschini*, Genady Ya. Grabarnik†, Filippo Poltronieri‡, Domenico Scotece*, Larisa Shwartz§, Cesare Stefanelli‡, Mauro Tortonesi‡, Mattia Zaccarini‡
* University of Bologna, Bologna, Italy
Email: {davide.borsatti,walter.cerroni,luca.foschini,domenico.scotece}@unibo.it
† Department of Mathematics and Computer Science, St. John's University, Queens, NY, USA
Email: grabarng@stjohns.edu
§ Operational Innovation, IBM TJ Watson Research Center, NY, USA
Email: lshwart@us.ibm.com
¶ Distributed Systems Research Group, University of Ferrara, Ferrara, Italy
Email: {filippo.poltronieri,cesare.stefanelli,mauro.tortonesi,mattia.zaccarini}@unife.it

***Abstract*—Kubernetes provides several functions that can help service providers to deal with the management of complex container-based applications. However, most of these functions need a time-consuming and costly customization process to address service-specific requirements. The adoption of Digital Twin (DT) solutions can ease the configuration process by enabling the evaluation of multiple configurations and custom policies by means of simulation-based what-if scenario analysis. To facilitate this process, this paper proposes KubeTwin, a framework to enable the definition and evaluation of DTs of Kubernetes applications. Specifically, this work presents an innovative simulation-based inference approach to define accurate DT models for a Kubernetes environment. We experimentally validate the proposed solution by implementing a DT model of an image recognition application that we tested under different conditions to verify the accuracy of the DT model. The soundness of these results demonstrates the validity of the KubeTwin approach and calls for further investigation.**

***Index Terms*—Service Management and Orchestration, Kubernetes, Simulation, Optimization.**

I. Introduction

The management of large-scale microservice architectures requires powerful orchestration solutions such as Kubernetes, the predominant tool for this task. Kubernetes provides a plethora of functions to manage the different aspects of service provisioning, such as cluster creation and federation, automated replication, and updates management. These functions provide a toolbox for service managers who need to configure the behavior of Kubernetes applications and tune their performance.

Finding the most appropriate set of configurations for a Kubernetes application represents a challenging and compelling problem [1], [2], that requires evaluating a large set of potential configuration parameters and perhaps even considering the realization of custom resource management solutions (e.g., for smart scheduling of software components). For example, there is the need to evaluate adequate lower and upper bounds for the number of replicas to associate at each microservice to avoid over-scaling or over-provisioning. In turn, this evaluation process requires dealing with time-consuming and costly operations such as the implementation of custom testbeds or configuring complex cloud-based Kubernetes deployments. To fully benefit from the potential of orchestration tools such as Kubernetes, there is a need for solutions that can approximate the behavior of Kubernetes applications, thus allowing the evaluation of different configurations safely and quickly.

Digital Twin (DT) approaches could represent a compelling step forward for the resolution of this problem. In fact, DT approaches enable what-if scenario analysis [3], [4], thus implementing a faster (and parallelizable) process for the exploration of a larger number of configurations, and even allowing the rapid prototyping of custom Kubernetes functions (e.g., autoscaling, scheduling). As a result, DTs could be very effective in speeding up the parameter identification process, as well as in significantly broadening its scope, with potentially significant costs savings [5], [6].

However, *defining an accurate DT of a Kubernetes application is a challenging task*. Several variables need modelings, such as the average container processing time, the communication latency between the cluster's nodes, and internal Kubernetes control loops. These parameters can be defined a priori or collected from historical data through statistical analysis. It is important to note that the quality of these parameters influences the ability of the DT to accurately reenact the behavior of its Kubernetes application.

Among these variables, modeling the service time of each microservice arguably represents the most critical step in the realization of the DT. At the moment of this writing, there is no consolidated solution to model the statistical distribution of the response time of Kubernetes applications. We believe that a good statistical distribution of the service time is essential to have a simulation environment with a good approximation of the modeled application. To contribute to this field, we propose KubeTwin, a framework that enables the definition of DTs of Kubernetes applications. This paper analyzes the problem of parameter identification for creating an accurate statistical description of a Kubernetes application. We present a generically applicable solution, which can analyze a set of

observations/metrics collected from a Kubernetes application to create its DT model.

To demonstrate the viability of this solution, this paper presents the case study of a realistic image recognition application that we implemented and deployed on a Kubernetes testbed. Experimental results show how the proposed solution is capable of creating a DT model which can provide a good approximation of the real system even under different working conditions. We believe that the proposed framework is a valuable contribution towards the adoption of DTs for what-if scenario analysis which motivates us to further research.

II. Digital Twins of Kubernetes Applications

KubeTwin is a framework we realized to define a DT of a Kubernetes environment. With the term environment, this paper specifies the overall components behind a Kubernetes application, such as pods, computing resources, services (e.g., load balancers), and so on. Each component is a "per se DT" that coexists with other DTs of Kubernetes components in a larger Kubernetes environment. This concept is referred to as "composability", a foundational property of DT discussed by Minerva et al. in [7].

With KubeTwin we can reenact the whole Kubernetes environment at a very fine-grained level, simulating every service request from the moment of its generation to the delivery of the response.

The overall architecture and the KubeTwin main components are visible in Fig. 1. Specifically, computing nodes, called KTNodes, have resources with configurable characteristics, e.g., CPU or GPU cores, and can be of two different types: edge nodes, e.g., edge servers hosted in small-size data centers located close to the end-user premises, and medium- and large-size data centers located at Cloud facilities. KTNodes are logically divided into *clusters*, which group a set of KTNodes in a specific location.

A KTNode can execute multiple KTPods, each one hosting a single KTContainer. Within KubeTwin, a KTContainer is the single unit of execution, implementing a single software component describing the amount of CPU and memory required for its execution. Furthermore, KubeTwin associates with each KTContainer a statistical distribution modeling the response time of the microservice it is simulating.

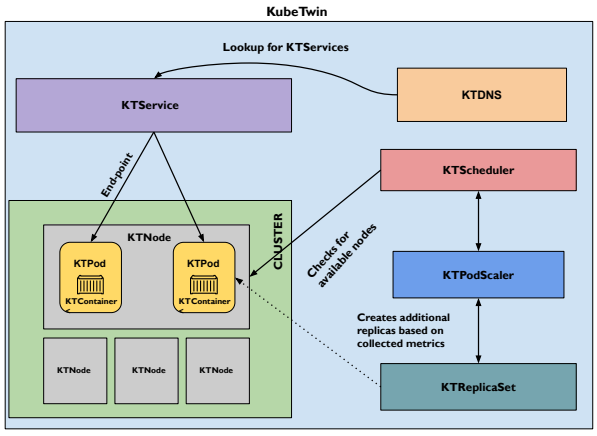

Fig. 1. The KubeTwin Architecture.

Finally, to simulate the request processing at each container level we adopt a G/G/1/FIFO queuing model. Specifically, KubeTwin users can configure several parameters, such as the maximum queue size and the service request arrival rates associated with a specific software component. For the latter, KubeTwin provides several choices of widely used distributions: exponential, log-normal, Pareto family, and others. In case of multiple replicas of the same service, we implement a queue before each container of the replica set. This choice is justified by the default load-balancing mechanism of Kubernetes, which redirects each request to a pod of the replica set with a given probability. Let us note that finding a suitable configuration for these parameters is important to define an accurate DT model, i.e., a model that can mimic the behavior of a Kubernetes environment with a good degree of accuracy.

Regarding the scheduling process, the KTScheduler is one of the main components for the management of computing resources according to a set of configurable policies, including the result of automated scaling procedures. Specifically, the KTScheduler is responsible for scheduling, i.e., allocating, KTPods onto KTNodes. This association is implemented via a filter-and-score procedure that analyzes the residual capacity of the pool of computing resources to assign them a score – according to a scheduling policy – and then selects the node with the highest score.

All the above components enable a fine-grained evaluation of a Kubernetes DT, thus allowing KubeTwin adopters to run what-if scenario analysis and evaluate the performance of their Kubernetes environments under different conditions (e.g., workload and computing changes). Following this approach, the outcomes of DT evaluations can be fed into custom policies and configurations for the real Kubernetes environment.

III. Model and Parameters Identification

As mentioned in the previous section, an accurate statistical description of the target application is necessary to have a reliable DT model of the Kubernetes environment. For this task this paper proposes an innovative *simulation-based inference procedure* [8] that identifies the configuration parameters described above. Specifically, the inference procedure explores the space of possible configuration parameters, comparing the metrics collected from a real-life Kubernetes application and the outcome of a DT simulation with a sample parameter set, and identifying the parameter set Σ that minimizes the statistical difference between those two observations. In other words, our procedure solves the following optimization problem:

$$\underset{\Theta,\Sigma}{\arg\min} \quad f(x,y) \tag{1}$$

where x represents a set of observations taken from the Kubernetes application and y is the observations taken from a simulation run using a model Θ with parameters Σ. The objective function $f(x,y)$ measures the statistical difference

between the probability density functions of x and y, thus indicating how distant the observations generated from the DT are from the ones generated from the Kubernetes application.

Simulation-based inference represents a relatively computationally intensive process, and in our case problem (1) typically has to deal with a relatively large search space to explore. However, in our experience, simulation-based inference approaches are far more accurate and robust than other approaches leveraging simplistic approximations, e.g., matching the mean processing times values of the target microservices.

Let us note that while we expect that the observations x come from metrics collected from a Kubernetes application, the observations y are generated by the DT with a model configuration $\Theta = m_1, m_2, \ldots, m_n$. Θ is a set containing n – where n is the number of microservices composing the application – random variables m_i, describing the processing time of the corresponding microservices.

As for the modeling of the random variable $m_i \in \Theta$, the simulation-based inference procedure provides several possibilities to describe their distributions. Therefore, users can choose the distribution that could better fit the microservices processing times: a log-normal distribution, a gaussian distribution, a Gaussian Mixture Model (GMM) with a configurable amount of components, and so on.

To solve problem (1), the optimizer looks for the configuration parameters $\sigma_i \in \Sigma$ of all random variables $m_i \in \Theta$ that allow the DT to produce observations y that are most statistically similar to x. This procedure allows the DT to predict the behavior of the Kubernetes application with higher precision. There are several possible methods to measure statistical similarity, including calculating the Kolmogorov–Smirnov statistics of two observations x and y [9], their Wasserstein distance [10], or their Wilcoxon-Mann-Withey distance [11]. The outcome of the simulation-based inference procedure will be the service time configurations for Θ to be used for reenacting the DT model of the Kubernetes application.

With regard to the problem complexity, it is worth noting that the type of distribution influences the solution of the optimization problem (1), i.e., increases or decreases the dimension of the solutions' search space. Specifically, finding a solution to the optimization problem (1) would require exploring a relatively large search space, which we can approximate using the following:

$$D \approx n \times \sum_{i=1}^{n} k_{m_i} \tag{2}$$

where n is the number of microservices and k_{m_i} the number of components chosen to describe the random variable m_i.

Concerning the optimization algorithm to implement the simulation-based inference procedure, we leverage the Quantum-inspired Particle Swarm Optimization (QPSO) algorithm of the ruby-mhl library (https://github.com/mtortonesi/ruby-mhl). In our experience, this metaheuristic approach strikes a good trade-off between convergence speed and capability to effectively explore large spaces in search for global minima [4].

IV. Use-Case Application

In this paper, we choose an image recognition application as a reference use case to be discussed. The image recognition service is an application that lets users identify objects in a picture, such as a photo captured by the user's phone or camera feeds' frames. When invoked, the application returns a message containing the names of the recognized objects to the user who requested the service. It is conceivable that this application would run on dedicated computing resources given its high computational requirements, thus allowing users to save the battery life of their equipment [12].

The image recognition app is a common example of an edge computing application, as it can process raw data produced by near cameras or Closed Circuit Television (CCTV) [13]. Processing the images directly at the edge of the network is beneficial in reducing network latency and usage since it avoids uploading collected image frames to the cloud for processing.

We built the image recognition app as a chain of two different microservices, the first implementing image resizing and the second implementing an object recognition algorithm. Both microservices leverage the Flask Python framework to allow network communication via REST-ful APIs. The first microservice of the chain (MS1) takes the image and normalizes it creating a representative array. Then MS1 sends this array to MS2, which classifies the image by leveraging a pre-trained model classifier. The final output of MS2 is the name of the object recognized from the input image. This information is then relayed back to the user through MS1.

Regarding the orchestration, we encapsulated MS1 and MS2 as two different containers allowing Kubernetes to manage and orchestrate them. Then, we configured a small-scale testbed composed of two Virtual Machines (VM) with four vCPU cores and 8GB of RAM each, running a standard installation of Kubernetes. To make the image recognition application available, we deploy the two microservices as two different containers running in separate pods. In addition, for each microservice, we define a Kubernetes NodePort Service for handling the services' name resolution, load balancing functions, and exposing the application endpoints. Finally, we defined a deployment for each microservice to manually scale the number of replicas of each component.

V. Experimental Results

The first step to define an accurate DT model for the image recognition application regards the metrics collection. Specifically, we are interested in estimating the processing time for MS1 and MS2 and the Time To Resolution (TTR), i.e., the total time to serve a generic request in a baseline scenario using the testbed deployment described above. The baseline configuration is obtained by analyzing the application behavior in a steady-state scenario, using one replica for each

microservice and a workload of non-overlapping requests (i.e., no queue time).

A. Parameters Inference

To collect these metrics, we developed a request generator using the Python programming language that can send requests with a configurable interarrival time to an HTTP endpoint. For the baseline scenario, we configured the request generator to send 2000 requests with a rate of one request per second (RPS). With this configuration, the interarrival time between subsequent requests is much larger than the average service time (around 50ms), thus canceling the requests' queuing time. In fact, minimizing the effect of queuing times is essential to properly capture the behavior of service processing times and to make sure the system under analysis is stable and operating in steady state conditions.

For each request, we collect the end-to-end Time To Resolution (TTR) that we measure as the time needed to complete an HTTP request to the image recognition application. In addition, we also collect the service time for MS1 and MS2 by collecting the time required to execute the instructions within each HTTP function from their logs.

Then we compute the Simple Moving Averages (SMA) with $n = 50$ of the processing times for all observations of MS1 and MS2. The behavior obtained by these calculations shows that MS1 and MS2 appear as stationary processes since their moving averages do not have high variations. Specifically, the SMA of MS1 is around 10 ms, while the one for MS2 is around 30 ms.

By assuming the stationarity, a G/M/1 queue model should describe well the service processing function of both microservices. Therefore, the simulation-based inference process described in Section III should be capable to find a DT model that well fits the image recognition application.

The first requirement for the fitting process is to define a KubeTwin configuration file containing the description of the two microservices, the available computing resources, and other Kubernetes-related configurations. For these experiments, we would fit the processing time of MS1 and MS2 using two GMM random variables with three components. Each component has a weight, a mean, and a sigma parameter for which we fix lower and upper bounds to help the optimization algorithm to converge.

Concerning the QPSO parameters, we set the number of iterations to 200, the swarm size to 50, and the α contraction-expansion parameter to 0.75 [4]. At each iteration, the optimizer evaluates 50 different configurations, from which it selects the population's best until it reaches the maximum number of iterations. As a function to measure the statistical distance between the outcome of the simulation and the metrics, we used the Wasserstein distance implemented in the SciPy Python Library.

We report the results of the simulation-based inference process in Fig. 2, which shows the Probability Density Functions generated using i) the image recognition application's log

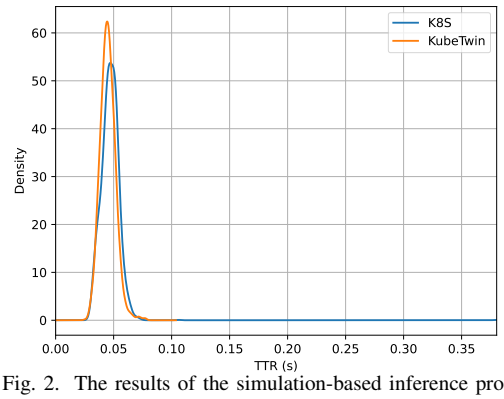

Fig. 2. The results of the simulation-based inference process.

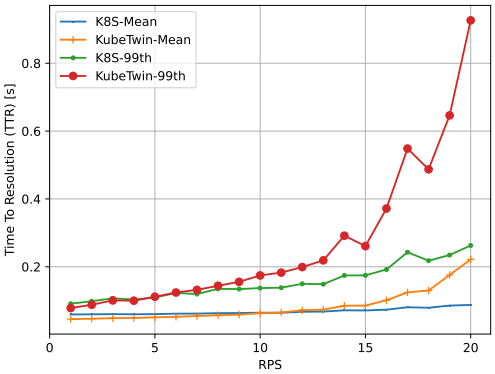

Fig. 3. The average and the 99th percentile TTR values collected for 5000 requests with KubeTwin and the K8S under different workload (RPS).

(K8S) and ii) the best configuration found by the simulation-based inference process (KubeTwin). We can observe how the proposed method can generate an accurate model for the DT that, when executed, can reflect the distribution of the end-to-end TTR values of the real image recognition application. Let us also note the distributions illustrated in Fig. 2 have a long right tail, thus indicating some high TTR values peaks for the image recognition application. We believe these peaks are the results of naming resolution, waiting times of containers, and other situations in which occurrences and distributions are difficult to model. However, these peaks are limited in number (1 or 2), thus not changing the steady-state behavior of the Kubernetes application. Overall, the results illustrated in Fig. 2 show that the proposed method can find an accurate DT model that mirrors the real Kubernetes application under the baseline scenario. However, the DT model should still provide good accuracy even when operating in different working conditions (e.g., higher requests load).

B. Validating the Model Under Different Conditions

To verify the accuracy of the DT model, we launch an experimental campaign to collect and compare the TTR values from 5000 requests for both the Kubernetes application and the Digital Twin under different workloads. The idea is to show that the DT model can well reproduce the behavior of the real application under a modified scenario, thus evaluating if the

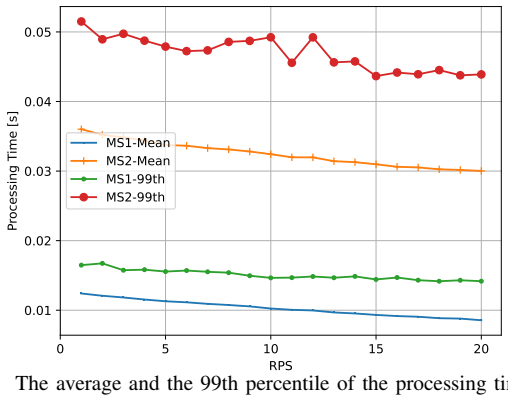

Fig. 4. The average and the 99th percentile of the processing time for MS1 and MS2 from RPS 1 to RPS 20.

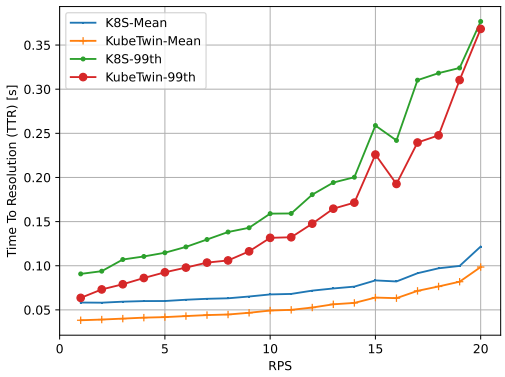

Fig. 5. The average and the 99th percentile TTR values collected for 5000 requests using the model Fitted at RPS 20.

fitted model can still be a good representation of the image recognition application.

In this regard, we built a Python application that sends a total amount of requests at different Requests Per Second (RPS) values. Let us note that generating a workload with a constant frequency (measured as the number of requests sent in a second) is a per se challenging task. Even if there are several tools available that can execute a stress test of a web-based application (e.g., the Apache benchmark tool), none of them allows the user to specify a target RPS value. In fact, the capacity to stress an application using a target RPS mainly depends on the computing resources available on the machine generating the requests.

For this validation, we collect for each experiment the end-to-end TTR values of all 5000 requests to compare and verify their distribution. Fig. 3 shows the Mean TTR (MTTR) and the 99th percentiles values on the y-axis that we collected from the execution of both the image recognition application (K8S) and its DT for all experiment (from 1 to 20 RPS) using the baseline deployment – one replica per microservice. By observing Fig. 3, we can verify how the MTTR values computed by the DT model are a good approximation of the image recognition application until around 15 RPS. Then, starting from 16 RPS, the MTTR values of the DT model diverge from the ones of the application.

Considering the high variance of the results, analyzing only the MTTR values could not be enough to understand the model performance. Therefore, Fig. 3 also shows the distribution of the 99th percentiles for the application and the DT, which we believe can provide a better insight. Analyzing the 99th percentile values, the divergence between the simulation and the application is even more evident. The differences revealed that the DT model was probably overestimating the service time of the microservices at higher RPS. To prove this behavior, we gather the logs from the microservices at different RPS in Fig. 4. From these results, we can see that the service time of each microservice decreases with the higher request rate. This speedup effect in the response time is probably caused by optimizations made at the CPU level (e.g., caching) that increase the performance of very active processes. Following this finding, we utilized the value obtained for RPS 20 to re-fit the model with the same procedure described in Section V-A. Then, we remade the measurement campaign and plotted the results in Fig. 5. As visible from the picture, the model is much more precise in forecasting the application response time, with a Mean Square Error (MSE) decrease of about $0,0013$ for the model fitted at one (Table I.)

TABLE I
MSE VALUES FOR THE MTTR AND THE 99TH PERCENTILE

Model	MSE - Mean	MSE - 99th
Baseline (1R)	0.001641	0.042060
RPS 20 (1R)	0.000364	0.001200
RPS 20 (3R)	0.010873	0.077834

C. Multi-Replicas Deployment

As a second validation, we want to verify if the DT model described in the previous section can also approximate the behavior of the Kubernetes application when the deployment increases in size. More specifically, we set the number of replicas to 3 for each microservice, and we collected the TTR logs from 5000 requests under different workloads from RPS 1 to RPS 40.

At the same time, we run the DT model using a 3 replicas configuration to compare the results. These are visible in Fig. 6, which illustrates how KubeTwin can provide a good estimate of the real application up to 40 RPS. From then, the two systems diverge, and the 99th percentile of the DT model reports a higher value when compared to the K8S application.

We are currently analyzing why the system remains stable even when the load increases. Of course, with short response times (i.e., around 40ms), the results are drastically impacted by numerous sources like network variability, CPU scheduling, and others.

VI. RELATED WORK

Kubernetes represents the de-facto orchestration solution to manage a wide range of container-based applications. Among the related efforts, authors in [14] present a theoretical optimization framework for edge-to-cloud offloading of Vehicle-to-everything (V2X) tasks and an edge-to-cloud offloading

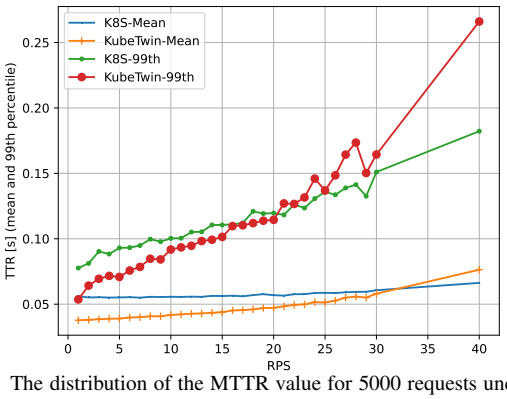

Fig. 6. The distribution of the MTTR value for 5000 requests under different RPS values using a 3 replicas deployment and the RPS 20 model.

decision algorithm called ECODA. Furthermore, in [15], the authors present a hybrid shared-state scheduling framework model that delegates most tasks to distributed scheduling agents. Watanabe et al. propose a Kubernetes-based prototype of a Multi-access Edge Computing architecture to which users can offload applications [16]. The authors in [17] also investigate Kubernetes scheduling techniques to find the main gaps, related challenges, and possible improvements to the current state-of-the-art. In [1], Santos et al. propose a network-aware scheduling approach for container-based applications that extends the default scheduling mechanisms of Kubernetes to consider the current network conditions.

Despite all these efforts, few initiatives are investigating DT of Kubernetes applications. Differently, this work takes the DT perspective and presents a novel framework to define DTs of Kubernetes applications for running what-if scenario analysis. We believe this work can bring many contributions to researchers and practitioners investigating custom and adaptive scheduling techniques.

VII. Conclusions

Digital Twins (DTs) of Kubernetes applications can simplify the configuration problems with simulations and what-if scenarios analyses. However, modeling accurate DTs is challenging and requires proper design and testing efforts.

To simplify this process, we present the KubeTwin framework, a novel tool that provides a simulation-based approach to the definition and evaluation of DTs. This work focuses on the simulation-based inference procedure we devised to create an accurate DT model leveraging a set of observations collected from the real application. We experimentally validated the proposed solution to show its viability by discussing and presenting a use case of an image recognition application. The obtained results push for further investigation into the characterization of microservices processing to create even more accurate and load-adaptive models.

Acknowledgements

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRPP) of Next Generation EU (NGEU), partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"), and by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - NGEU.

References

[1] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 351–359.

[2] T. Kiss, J. DesLauriers, G. Gesmier, G. Terstyanszky, G. Pierantoni, O. A. Oun, S. J. Taylor, A. Anagnostou, and J. Kovacs, "A cloud-agnostic queuing system to support the implementation of deadline-based application execution policies," *Future Generation Computer Systems*, vol. 101, pp. 99 – 111, 2019.

[3] M. Mendula, A. Bujari, L. Foschini, and P. Bellavista, "A data-driven digital twin for urban activity monitoring," in *IEEE Symposium on Computers and Communications (ISCC)*, 2022, pp. 1–6.

[4] W. Cerroni, L. Foschini, G. Y. Grabarnik, F. Poltronieri, L. Shwartz, C. Stefanelli, and M. Tortonesi, "Bdmaas+: Business-driven and simulation-based optimization of it services in the hybrid cloud," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 322–337, 2022.

[5] G. Nardini and G. Stea, "Using network simulators as digital twins of 5G/B5G mobile networks," in *IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022, pp. 584–589.

[6] D. Van Huynh, V.-D. Nguyen, V. Sharma, O. A. Dobre, and T. Q. Duong, "Digital twin empowered ultra-reliable and low-latency communications-based edge networks in industrial iot environment," in *ICC - IEEE International Conference on Communications*, 2022, pp. 5651–5656.

[7] R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, 2020.

[8] G. Avecilla, J. N. Chuong, F. Li, G. Sherlock, D. Gresham, and Y. Ram, "Neural networks enable efficient and accurate simulation-based inference of evolutionary parameters from adaptation dynamics," *PLoS Biology*, vol. 20, no. 5, p. e3001633, 2022.

[9] R. Legin, Y. Hezaveh, L. P. Levasseur, and B. Wandelt, "Simulation-based inference of strong gravitational lensing parameters," *arXiv preprint arXiv:2112.05278*, 2021.

[10] M. Sommerfeld and A. Munk, "Inference for empirical wasserstein distances on finite spaces," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 80, no. 1, pp. 219–238, 2018.

[11] C. Bartolini, C. Stefanelli, and M. Tortonesi, "Modeling it support organizations using multiple-priority queues," in *IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 377–384.

[12] B. Herlicq, A. Khichane, and I. Fajjari, "Nextgenemo: an efficient provisioning of edge-native applications," in *ICC - IEEE International Conference on Communications*, 2022, pp. 1924–1929.

[13] Y. Chen, S. Zhang, Y. Jin, Z. Qian, and S. Lu, "Multi-server multi-user game at edges for heterogeneous video analytics," in *ICC - IEEE International Conference on Communications*, 2022, pp. 841–846.

[14] E. C. Cejudo and M. Shuaib Siddiqui, "An optimization framework for edge-to-cloud offloading of kubernetes pods in v2x scenarios," in *IEEE Globecom Workshops (GC Wkshps)*, 2021, pp. 1–6.

[15] O.-M. Ungureanu, C. Vlădeanu, and R. Kooij, "Kubernetes cluster optimization using hybrid shared-state scheduling framework," in *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, ser. ICFNDS '19. New York, NY, USA: Association for Computing Machinery, 2019.

[16] H. Watanabe, R. Yasumori, T. Kondo, K. Kumakura, K. Maesako, L. Zhang, Y. Inagaki, and F. Teraoka, "Contmec: An architecture of multi-access edge computing for offloading container-based mobile applications," in *ICC - IEEE International Conference on Communications*, 2022, pp. 3647–3653.

[17] C. Carrión, "Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges," *ACM Computing Surveys*, vol. 55, no. 7, Dec 2022.