LQ-GNN: A Graph Neural Network Model for Response Time Prediction of Microservice-based Applications in the Computing Continuum

Matías Richart, Juan-Luis Gorricho, Javier Baliosian, Luis M. Contreras, Alejandro Muñiz and Joan Serrat

***Abstract*—To address the challenges posed by the deployment of microservices of future end-user applications in the cloud continuum, a performance prediction model working together with a network elasticity controller will be needed. With that aim, this work introduces Layered Queuing-Graph Neural Networks (LQ-GNN), a novel Machine Learning (ML) approach to develop a generalized performance prediction model for microservice-based applications.**

Unlike previous works focused on individual applications, our proposal aims for a versatile model applicable to any microservice-based application, integrating the Layered Queueing Network (LQN) modeling with Graph Neural Networks (GNN). LQ-GNN allows to efficiently estimate the response time of applications under different resource allocations and placements on the computing continuum. The obtained evaluation results indicate that the proposed model achieves a prediction error below 10% when considering different evaluation scenarios.

Compared to existing methodologies, our approach balances prediction accuracy and computational efficiency, making it viable for real-time deployments. Consequently, ML-based performance prediction can significantly enhance the resource management and elasticity control of microservice-based architectures, leading to more resilient and efficient systems.

***Index Terms*—Computing Continuum, Elasticity, Microservice-based applications, Graph Neural Networks, Machine Learning.**

I. Introduction

IN the last decade, more and more computing tasks have been shifted to the cloud, driven by the advantages of resource flexibility for users and CapEx (Capital Expenditures) efficiency for both users and service providers [1]. Users benefit from the ability to scale their demand for resources as needed, paying only for usage time, while cloud operators maximize their cost efficiency by sharing their infrastructure among multiple users. In recent years, fog and edge computing have emerged as extensions of cloud computing, getting closer to data sources and establishing a seamless *computing continuum* [2].

The computing continuum facilitates the deployment of novel services across a distributed infrastructure, catering to various applications such as autonomous vehicles, smart cities, or content delivery. These services exhibit diverse requirements and are often unlikely to be supported only by traditional remote cloud computing. For instance, they may need low-latency connections for a rapid decision-making regarding their input data sources and a significant computing capacity for intricate data analysis. The computing continuum solution offers a wide range of computational and communication resources, promoting more effective support of heterogeneous demands.

On the other hand, recently, there has been a significant adoption of the so-called multi-tier microservice architecture as an alternative software development framework for cloud applications [3]. This new paradigm moves away from the conventional monolithic application, where a single service encapsulates all functionality, towards a model where individual microservices provide fine-grained, single-concern, and loosely-coupled services. This architecture enables building more sophisticated functionalities [4] depending on the chosen microservices composition.

Microservices are intended to be deployed on the cloud continuum to run latency-sensitive applications that interact directly with their end-users [5]. In this sense, availability and response time metrics reflect the user's experience and are commonly established as service-level objectives (SLOs). Particularly for microservice-based applications, the response time SLO (also named application latency in the literature) holds significant importance, and to meet this objective, many operators have traditionally allocated some excess resources on their cloud platforms. However, this practice incurs a considerable cost, where commonly 20% of the computing cost is due to over-provisioning [6]. Hence, even minor enhancements in cloud resource allocation could lead to substantial cost savings at scale.

In the above scenario, to preserve an efficient use of resources, it is mandatory to satisfactorily solve the problem of resource allocation. The aim is to minimize the assigned resources while meeting all QoS requirements. Moreover, since applications workloads are time-variant, it is desirable to dynamically adjust the allocated resources (such as CPU, RAM, or even new microservice instances) in real-time. The approach of dynamically, autonomously and optimally adapting to a varying workload is known as *elasticity*, and it is generally implemented by a so-called *elasticity controller* [7].

Unlike monolithic deployments, microservices promote a more accurate resource allocation by scaling individual ap-

Matías Richart and Javier Baliosian are with the Computer Science Department, School of Engineering, University of the Republic, Montevideo, Uruguay (e-mail: mrichart@fing.edu.uy; baliosian@fing.edu.uy).

Matías Richart, Juan-Luis Gorricho and Joan Serrat are with the with the Department of Network Engineering, Polytechnic University of Catalonia, Barcelona, Spain (e-mail: juan.serrat@upc.edu; juan.luis.gorricho@upc.edu).

Alejandro Muñiz and Luis M. Contreras are with Telefónica CTIO, Madrid, Spain (e-mail: alejandro.muniz@telefonica.com; luismiguel.contrerasmurillo@telefonica.com).

plication components. However, this fine granularity presents unique challenges to implement the elasticity concept, different from those of conventional Service Oriented Architecture (SOA) applications [8]. Microservices add complexity to resource management, as the elasticity controller must consider the dependencies between microservices to ensure the required quality of service (see Section II for more details). In addition, given the implications of dependencies, microservice-based applications may experience some specific performance phenomena when the workload changes very quickly [9].

To overcome these challenges and effectively implement an elasticity approach, it is necessary to conceive a model to predict the performance of different potential resource allocation and placement solutions for the constituent microservices of the applications. In this regard, Machine Learning (ML) approaches have emerged as a powerful modeling and prediction tool [9]–[12]. However, despite recent advances in machine learning to solve this problem, a fundamental limitation of existing works comes from the resultant learned models, as they are specific to the application used for training. In addition, the above-mentioned ML-based works are based on learning specific relationships, like the traffic load with the application response time, instead of the dependencies between microservices.

In this paper, we seek to overcome these limitations and build a more generic performance predictor, as a necessary tool for any microservice-based application deployment and placement, as an effective enabler of elasticity in the cloud continuum. The contributions made by this work can be summarized as follows: First, we adopt the Layered Queueing Networks (LQNs) [13] technique to build an enriched graph model for applications based on microservices. Second, we adapt the LQN resultant graph of any application to be used as the input of a Graph Neural Network (GNN) to predict the application deployment's performance given the allocated resources and placement of the corresponding microservices. Finally, we develop a testing environment to assess the accuracy of our performance predictions, comparing the results of the GNN predictions with those obtained by means of a conveniently adapted state-of-the-art simulator. The proposed predictor could then be used as part of an elasticity controller to decide the best deployment option for the application, although this is out of the scope of this work.

II. Challenges for Performance Prediction of Microservice-Based Applications

Performance prediction for resource allocation and elasticity of microservice-based applications presents significant challenges, primarily due to the distributed and interdependent nature of their constituent microservices. Unlike traditional monolithic systems, microservices require the elasticity controller to manage resource allocation across multiple loosely coupled services while ensuring end-to-end Quality of Service (QoS). This complexity arises from the dependencies between microservices and their placement across a computing continuum, creating several performance prediction challenges.

One key challenge is the variability of execution paths for different types of requests. For instance, in the Social Network application proposed in [8], requests such as reading a user's timeline or viewing another user's posts follow different ordered sequences of microservices. Figure 1 shows part of the application's architecture and illustrates the distinctive paths taken by different types of requests. Complementarily, some microservices handle multiple request types (e.g., Post Storage), while others are specific to particular paths (e.g., Home Timeline). Moreover, even for a single request type, different sub-paths may be followed based on data availability, leading to varying workloads across microservices.

Performance prediction becomes even more difficult due to backpressure effects. This occurs when a microservice experiences an increased response time not because of its own resource saturation, but due to congestion in downstream services. For example, if the Post Storage MongoDB becomes slow, requests will pile up at the Post Storage service, inflating the response time despite adequate resource availability. This makes it challenging to identify the root cause of QoS violations, as neither resource utilization nor response time alone can accurately indicate where performance issues originate.

Another major issue is the cascading effect [9], where sequential scaling causes delays in accommodating an increased load. When a bottleneck microservice scales up, the resultant load increase propagates downstream, potentially requiring subsequent microservices to scale up as well. This step-by-step adjustment delays the system's overall response to traffic spikes, worsening performance degradation during peak demand periods.

The combination of backpressure and cascading effects exacerbates the difficulty in pinpointing performance bottlenecks and the intended timely scaling. Mistaking the source of degradation can lead to delayed resource allocations, allowing queues to grow and consequently prolonging the recovery time.

Finally, the distributed deployment of microservices across diverse physical nodes introduces network strain, particularly under high load. Reliance on RPC or HTTP communications means that network delays can significantly contribute to tail response time, especially as queues grow inside network interface controllers. Thus, transmission bandwidths and delays must be considered for resource allocation and microservice placement.

Traditional autoscaling approaches like AWS or Kubernetes [14], [15] monitor and scale applications as single units, which is insufficient for microservice-based systems with strict performance guarantees. A more holistic view for performance monitoring, prediction and elasticity is needed—one that proactively considers microservice dependencies, workload changes, and network infrastructure to prevent performance degradation and ensure end-to-end QoS.

III. Related Work

As mentioned above, calculating the end-to-end response time of a microservice-based application is not straightforward. Each microservice exhibits distinct response time patterns, and the connections between them can be quite complex. Not only is the end-to-end response time of the request the

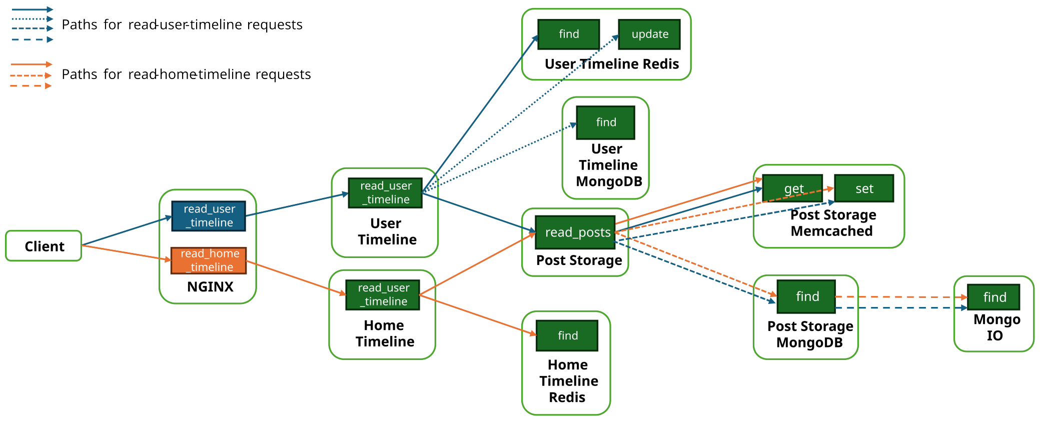

Fig. 1. Different possible requests' paths in the Social Network application.

result of multiple additions and maximum operations involving the response time of each microservice, but some microservices' response times are also influenced by the performance of neighboring microservices [9].

Therefore, a significant challenge arises: response time measures can only occur after setting a resource configuration to the actual infrastructure. Real-time experimentation of various alternatives is infeasible due to the potential impact on microservice performance when altering resources. Additionally, the search space for possible combinations grows exponentially when ranging from tens to hundreds of microservices within an application. To address this issue, recent research on the subject has focused on different alternatives to model the problem in a way that provides fast predictions on the performance of a given resource allocation. In this regard, research can be classified into three main strategies: model-based, simulation-based, and data-based.

The model-based approach involves developing a theoretical model of the application's behavior to analyze its performance. Traditional queueing theory is not enough to model the behavior of a microservice-based application with blocking connections between microservices. Therefore, an extended form of queueing called layered queueing is necessary. When a given software component calls another component and waits (blocked) for its reply, we have an example of layered queueing. For this, current research uses the Layered Queueing Network (LQN) model [13], [16]

The layered queueing model was introduced several years ago to model the behavior of distributed systems and has evolved over the years. LQN has become a standard for analytically modeling traditional distributed systems, with existing solvers and simulators designed to compute solutions [16]. It has also been applied to modeling the performance of cloud applications. However, it has not been until recently that the idea of using LQN to model microservice-based applications has emerged.

In this regard, ATOM is proposed in [17], an autoscaling controller for microservice-based applications that uses LQN models to estimate performance. The proposed controller works iteratively by the following steps: from monitoring data, an LQN model of the application is obtained, an LQN analytical solver is used to obtain performance estimates from the LQN model, and these estimates are used in a meta-heuristic algorithm to find a scaling solution; finally, the obtained solution is applied to the system. Regarding the estimate, the proposal is evaluated for a single application, and not much detail on the accuracy of the solver is provided.

A similar approach, which also takes advantage of LQN modeling for resource scaling, is presented in [18]. In this case, the LQN model is transformed to a *fluid approximation*, using a compact system of coupled ordinary differential equations (ODEs). Then, the resource allocation problem is formulated as a nonlinear optimization problem where the ODEs of the fluid approximation of the LQN are used as constraints. Finally, the optimization is solved with a series of relaxations such as local optimization, smooth approximations for linearization and differentiability, and transforming integer variables into continuous ones. The evaluation is performed for a single microservice-based application. The results of the performance estimates show an error of less than 8% for the estimate on the response time of microservices.

Despite the potential of modeling techniques, significant limitations arise when addressing the elasticity demands in microservices. Solvers often stumble because they rely on assumptions like Poisson arrivals, which do not align with the heavy-tail, autocorrelated nature of Internet traffic [19].

Alternatively, simulators [20] can model the performance of microservice-based application deployments more accurately for non-homogeneous traffic. However, running an appropriate simulation demands a significant computational load, turning into long run times. Moreover, the cost of running a simulation scales exponentially with the number of simulated incoming requests to be processed and the number of involved microservices. Consequently, this limitation severely restricts its utility when considering a real-time elasticity scenario where strict time constraints apply.

In this context, data-driven approaches using Machine Learning (ML) have recently been proposed. These approaches sidestep these pitfalls by deriving models directly from real-world data, capturing complex, non-linear dynamics, with impressive precision.

In [10] a resource manager called Sinan is proposed, which uses a Convolutional Neural Network (CNN) model for detailed short-term performance prediction and a Boosted Trees model that evaluates the long-term performance evolution. Sinan leverages a dual-model approach to achieve more accurate resource allocation predictions and to better account for system inertia when building up queues. By examining near-future outcomes and system behavior over time, Sinan dynamically adjusts per-tier resources online based on the service's runtime status and end-to-end quality of service (QoS) targets. However, its main limitation is that CNN is highly dependent on the graph of the application's microservices, and a different CNN is needed for each application.

Therefore, GNNs have recently become a go-to for microservice-based applications, thanks to their affinity for graph data structures and their possibility of being independent of the input graph structure. Moreover, by representing the application as a graph of microservices, GNN models can better capture the complex interactions, which can help in detecting back-pressure and cascading effects, since they understand the graph's dependencies [21].

In this sense, GRAF, as referenced in [9], employs a GNN to estimate the end-to-end tail response time of an application using the execution states of the microservices. Initially, it examines the workloads of the running application, which are combined with the CPU quota of the microservices to depict the node state of the graph of microservices. The input graph is constructed from the previous analysis and consists of a Direct Acyclic Graph (DAG) representing the connections between microservices. Subsequently, GRAF applies a gradient descent algorithm to identify the least resource configuration that meets the tail response time Service Level Objective (SLO). In the loss function, the fully trained GNN model is used to detect potential violations of the response time SLO. Regarding the estimated performance of the GNN, the authors report percentage errors between 20% and 30% depending on the response time ranges.

PERT-GNN follows a similar approach [11], a generic framework based on GNNs that predicts the end-to-end response time for microservice-based applications. It defines the interactions or dependencies of constituent microservices, which are observed from previous execution traces of the application, using the Program Evaluation and Review Technique (PERT). A GNN is built based on the PERT Graphs generated, and the task of response time prediction is formulated as a supervised graph regression problem using the graph transformer method. This approach is similar to the one used by GRAF, but it incorporates PERT Graphs and the graph transformer method. The authors show that using a PERT Graph improves response time predictions as more insight into the temporal dynamics of microservices is obtained. In this case, the prediction approach is evaluated for two different applications, obtaining a percentage error of about 12% with an absolute error of 1.2ms to 1.6ms.

Graph-PHPA, as referenced in [12], is another machine learning-based approach that employs a two-stage prediction method. This method utilizes both Long Short-term Memory (LSTM) and a GNN. The LSTM is used to predict the workload, while the GNN is used to model the relationship between the workload and the resource consumption of different microservices within the network. However, it should be noted that there is not much information available on how the graph is constructed using this approach. In this case, the node state is solely based on the resource consumption of each microservice. This approach differs from the previous ones as it combines LSTM and GNN and focuses only on resource consumption for the node state.

Our approach diverges from previous works on GNNs by aiming not at developing a model for a single application but at constructing a generalized model capable of predicting the performance of any microservice-based application. By integrating LQN modeling and considering microservice features such as the processing delay of requests, we leverage the intrinsic interactions among microservices to achieve a general model. Moreover, our model explicitly incorporates network latency between microservices as a key factor. By supporting different placement strategies across the computing continuum, it allows microservices to be deployed at any point within this infrastructure. This flexibility adds complexity to the problem, as the varying network delays between microservices become an essential consideration in the model's predictions.

IV. LQN-Based Graph Model

As stated in Section II, predicting the end-to-end response time of requests for microservice-based applications is a challenging matter. Therefore, it is critical to design a model that can replicate the microservices' behavior and interactions. In this regard, previous works have proposed different graph models; the most used ones are *Call Graphs*, these are Directed Acyclic Graphs (DAGs) where the nodes stand for microservices and the edges stand for the communication between them.

However, the Call Graph approach falls short of modeling all the complexities of microservice-based applications. As stated in [11], the main limitations of a Call Graph are:

- Lack of distinction between different processing paths: a call graph typically represents the entire application's call structure without distinguishing between the possible distinct processing paths followed by different requests.
- No representation of the internal microservice behavior: a call graph only captures the interactions between microservices without providing details about the internal stages or processing steps within each microservice.
- Inability to capture temporal dynamics: when a microservice has two or more outgoing links to other microservices, a call graph does not provide information about the order in which those subsequent processing requests are carried out, which can be essential for understanding the system performance and its behavior.

Hence, we propose building a graph model based on the Layered Queueing Network (LQN) technique. This technique allows us to represent not only the relationships between microservices, but also the different possible paths traversed by any request within the graph, as well as the behavior and performance of the involved microservices.

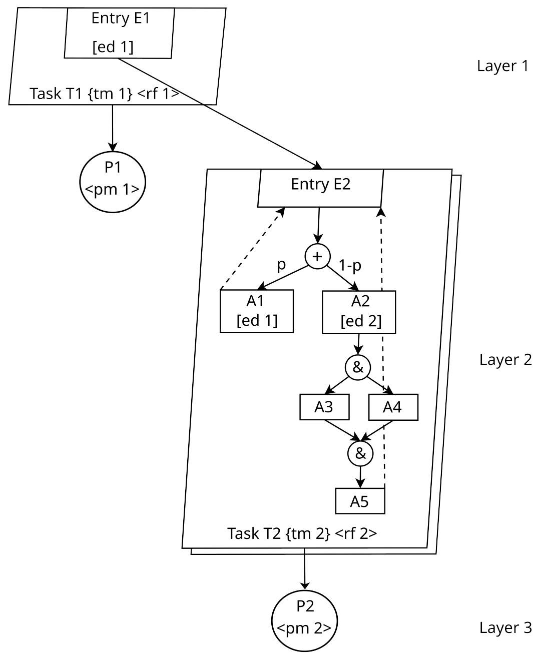

Fig. 2. LQN model example.

A. Layered Queueing Networks

First, let us briefly introduce the Layered Queueing Networks (LQNs) modeling technique. By extending traditional queueing networks, LQNs incorporate various layers of resources and services that interact with one another. Each layer contains different entities, which may represent a software module or a hardware resource, making requests to entities in lower layers. This method is particularly effective for systems with nested or hierarchical service requests, such as distributed systems or microservice-based applications. It allows performance metrics such as response time, throughput, and resource utilization to be obtained.

Within an LQN, the entities representing software modules are referred to as *tasks*, which possess queues and provide different types of services known as *entries*. If necessary, the behavior of each type of service can be modeled through *activities*. Figure 2 shows an example of an application modeled by an LQN. In LQN notation, a task is represented by a parallelogram, which encompasses additional parallelograms for its entries and rectangles for its activities. The *replication factor*, shown in angular brackets, indicates how many identical replicas are considered for each task. *Task multiplicity*, displayed in curly brackets, refers to the number of software threads available per task replica. Hardware resources are modeled by *processors*, illustrated as circles, and each task must be linked to one. The processor symbolizes the hardware computational units responsible for executing each task thread. *Processor multiplicity*, also shown in angular brackets, denotes the processor's level of parallelism.

Each entry has an *execution demand*, indicated in square brackets, representing the mean time it takes for that entry to process a request (we also call it *processing delay* in this work). However, if necessary, an entry behavior can also be modeled by a graph of activities (as in Task 2 of Figure 2), each of them with its own execution demand. Activities represent the finest level of detail in the model and are interconnected in a directed graph to denote the execution order. The model allows the requests to diverge into several sub-paths that eventually converge. The fork can be an AND (&), implying that all activities following the divergent point can execute concurrently, or an OR (+), which selects one of the sub-paths based on a given probability p.

Entries and activities are connected to other lower-layer entries through directed links, symbolizing *service requests* to lower layers. A request from an entry or an activity to another entry may yield a response to the requester; in this case, we consider this a synchronous request. The *service time* (or *response time*) for an entry to a request is the total time a single request holds it. This encompasses its own processing delay and any periods where it is blocked, waiting for its processor or for nested lower-level services to finish their tasks. This concept is the main difference from traditional queuing networks and is essential for modeling microservice-based applications.

B. LQN-Based Microservices Graph Model

The previously described LQN model represents an application execution very similar to a microservice-based architecture where the tasks correspond to the different microservices of the application. However, it is not a graph that can be directly considered an input to a GNN. Therefore, we propose a new graph representation for a microservice-based architecture by combining the LQN ideas with the traditional DAG representation of microservice-based applications.

In this regard, two main characteristics of LQNs need to be adapted. First, in LQNs, tasks include entries, which also include activities. Then, we propose a representation where each entity of the LQN model (tasks, entries, and activities) is a node of a graph and where the entries or activities of a task are connected to the corresponding task through a link. Given that in our scenario, we do not envision sharing computing resources between microservices (each microservice is allocated its own resources), we do not add a representation for processors (as in LQN) to our model. Instead, information regarding the resources allocated to a task is considered an attribute of the task node.

Secondly, the activities' graph allows forks, which implies that a request can follow different paths depending on the probability of the OR forks or even parallel paths in the case of an AND. We propose to model each path as a node of the graph that is connected to all the activities it traverses. Hence, forks in LQN would generate two separate paths in our model. This way, we have a detailed representation of a request's possible paths and its corresponding performances. Moreover, we substitute the modeling of the entries with the modeling of the paths and the activities that constitute them.

Following the LQN model, every activity processes requests and consumes time on the processors assigned to the respective microservice. An activity can also send a request to another activity of its own microservice or to a different microservice. Then, activities are also connected through links, following this path. In our model, we are always considering *synchronous* requests, which means that the execution of the requester waits while the requestee processes the request and generates the corresponding reply. However, we differentiate *blocking* and *non-blocking* calls, which determine if the thread of execution is blocked while waiting for the response or if the call is implemented in a way that does not block the execution.

Given that microservices can be deployed and distributed among possibly distant computing nodes in the computing continuum, we must also model the underlying network delay. In our modeling, the network delay between two microservices is modeled as a *dummy* task, which does not consume resources but has an activity with a processing delay equal to the network delay.

The rationale of the proposed representation is the following:

- The state of an activity (e.g., response time, throughput) depends on the state of the microservice it belongs to (e.g., assigned resources and resource utilization), on the state of the activities that send requests to it (e.g., throughput), on the state of the activities which it sends a request to (e.g., response time) and on all the paths that traverse the activity (e.g., amount of requests).
- The state of a microservice (e.g., load) depends on the state of all its activities (e.g., load, throughput).
- The state of a path (e.g., response time) depends on the state (e.g., response time, load) of all the activities it traverses.

In summary, our model of an application consists of a graph with three types of nodes: task, activity, and path. The task nodes represent the microservices and are connected to activity nodes. An activity node represents the minimum computation unit and is connected to other activities to form a sequence of activities a request can traverse. Additionally, all the activities of a request path are connected to a path node, which gathers information about the request path.

In Figure 3, we show a graph example for an application with four microservices (red nodes), each of them with a different amount of activities (green nodes). There are also two paths (light and dark blue nodes): P1, which is a sequence of three activities from T1 and T2, and P2, which is a more extended sequence that includes all the activities of the application.

C. Graph Construction and Processing-Delay Estimation

Deriving the complete LQN graph previously proposed involves two main tasks: (1) constructing the LQN graph, which consists of tasks, activities, and their connections, and (2) estimating the processing delay metrics for each task and activity.

For the first task, the LQN graph can be constructed using information provided by the application owner, such as a UML diagram. Some existing works propose automated approaches to derive LQNs from UML descriptions [22], [23], which could further facilitate this process. Additionally, it is necessary to extract request paths from the original application,

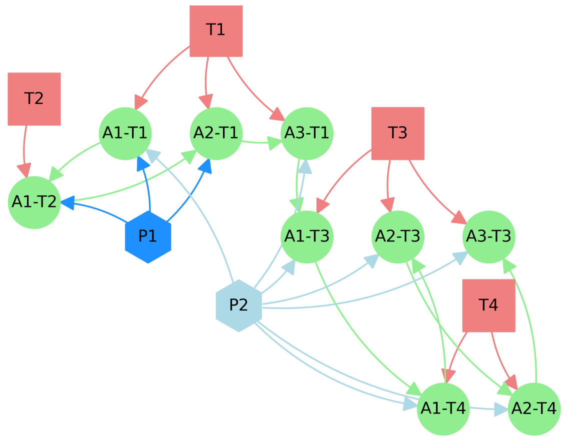

Fig. 3. Example for the proposed LQN-based graph model.

determine the load for each path, and gather information on microservice activities, such as request processing delays. In real-world scenarios, application performance monitoring tools [24] can be used to derive communication patterns, performance characteristics, and relevant metrics.

For the second task, estimating processing delays can be achieved by analyzing response time measurements of microservices and applying statistical estimation techniques. One possible approach is the maximum likelihood estimation method proposed in [25], which has demonstrated high accuracy in previous evaluations. This allows for a reliable assessment of processing delays without requiring extensive manual intervention.

However, in our study, this estimation process is not required as we rely solely on simulations. Since we have full control over the simulated environment, the processing delays are predefined within the application configuration. Therefore, while we highlight an approach that could be used in practical deployments, our experiments do not depend on this estimation technique.

V. LQ-GNN: Model Overview and Applications

The graph model proposed above defines a circular dependency between microservices, activities, and paths. To solve the circular dependencies, and inspired by the work from [26], we implement a Graph Neural Network architecture [1] consisting of a three-stage message passing algorithm that combines the states of tasks, activities, and paths and updates them iteratively. Finally, it combines these states to estimate the application's response time. We introduce this novel integration of Layered Queueing Networks (LQN) and GNNs as LQ-GNN.

A. LQ-GNN Design

In Algorithm 1, we present the proposed message-passing architecture in detail. First, hidden states h_k, h_a, and h_p are initialized using the initial features x_k, x_a, and x_p into

[1] A comprehensive description of how GNNs work can be found in [27].

fixed-size vectors representing feature embeddings. The initial features of tasks x_k are defined as a n-element vector that characterizes the microservice. In our case, this vector includes the number of threads configured in the microservice, the number of processor cores assigned, and the number of replicas deployed. We set the initial features of activities x_a with the processing delay of a request and with a boolean parameter, which indicates if the activity calls are blocking or non-blocking. Finally, the initial feature of paths x_p consists of the load on that path (as the average requests per second).

Once all hidden states are initialized, the message-passing phase starts. This phase is executed for T iterations, where T is a configurable parameter of the model. Each message-passing iteration is divided into three stages, representing the message exchanges and updates of the hidden states of tasks, activities, and paths.

In the first stage (from lines 11 to 17), the hidden state of each path is updated. For this, each activity connected to a path generates a message from its hidden state. Note that the only neighbors of a path are its activities. Then, all these messages are aggregated in the path in an ordered vector. Finally, the hidden state of the path is updated using the previous aggregation as input to a Gate Recurrent Unit (GRU) network.

For the second stage, a similar approach is followed to update the state of the activities. In this case, we have three different types of neighbors for each activity in the graph: the task to which the activity belongs, the paths that traverse that activity, and the predecessor and successor activities. Each of these types of neighbors generates a message to the activity. The task (line 19) and the paths (lines 20 to 22) send their hidden states directly. For the neighbor activities (lines 23 to 25), the messages are generated using a neural network that receives as entry the hidden state of source and destination activities and the direction of the connection (if the source is a successor or predecessor). Then, all these three types of messages $(m_{a,k}, m_{a,k}, m_{a,k})$ are aggregated using an edge-attention mechanism (line 26).

In the third stage, the state of each task is updated by combining the messages from all the activities that belong to that task. For this, each activity connected to the task generates a message from its hidden state. Then, all these messages are aggregated in an ordered vector, and the hidden state of the task is updated using the previous aggregation as input to a GRU network.

Finally, the updated hidden states are used to predict the application's response time. For this, we propose and evaluate two different predictions:

- the response time of each task (line 37)
- the response time of the entire application (line 39).

The functions R represent a readout function consisting of a neural network. For the prediction of tasks, this function is individually applied to all tasks' hidden states. For the application's response time, a sum pooling function is used to group the states of the paths, and then the readout function is applied to obtain one prediction.

Figure 4 shows the message-passing proposal over an example graph. In the figure, the yellow arrows represent messages received by activity A1-T2 from paths, tasks, and other activities; the green arrows represent messages received by path P1 from the corresponding activities, and the red arrow represent messages from activities to task T2.

Algorithm 1 Proposed GNN architecture.

```
1:  for each $k \in Tasks$ do
2:      $h_k^0 \leftarrow x_k$                ▷ [#threads, #cores, #replicas]
3:  end for
4:  for each $a \in Activities$ do
5:      $h_a^0 \leftarrow x_a$                ▷ [proc delay, blocking]
6:  end for
7:  for each $p \in Paths$ do
8:      $h_p^0 \leftarrow x_p$                ▷ [load]
9:  end for
10: for t=0 to T-1 do
11:     for each $p \in Paths$ do
12:         for each $a \in Activities(p)$ do
13:             $m_{p,a}^t \leftarrow h_a^t$
14:         end for
15:         $M_p^t \leftarrow [m_{p,1}^t, ..., m_{p,n_p}^t]$
16:         $h_p^{t+1} \leftarrow GRU(h_p^t, M_p^t)$
17:     end for
18:     for each $a \in Activities$ do
19:         $m_{a,k}^t \leftarrow h_k^t \mid k \in Tasks(a)$
20:         for each $p \in Paths(a)$ do
21:             $m_{a,p}^t \leftarrow h_p^t$
22:         end for
23:         for each $a' \in Activities(a)$ do
24:             $m_{a,a'}^t \leftarrow DNN(h_a^t, h_{a'}^t, dir_{a'})$
25:         end for
26:         $M_a^t \leftarrow \sum_{i \in N(a)} \gamma(m_{a,i}^t, h_a^t) * m_{a,i}^t \mid N(a) = Tasks(a) \cup Paths(a) \cup Activities(a)$
27:         $h_a^{t+1} \leftarrow GRU(h_a^t, M_a^t)$
28:     end for
29:     for each $k \in Tasks$ do
30:         for each $a \in Activities(k)$ do
31:             $m_{k,a}^t \leftarrow h_a^t$
32:         end for
33:         $M_k^t \leftarrow [m_{k,1}^t, ..., m_{k,n_k}^t]$
34:         $h_k^{t+1} \leftarrow GRU(h_k^t, M_k^t)$
35:     end for
36: end for
37: for each $k \in Tasks$ do
38:     $\hat{y}_k \leftarrow R_k(h_k^T)$
39: end for
40: $\hat{y}_a \leftarrow R_a(\sum_{p \in Paths} h_p^T)$
```

B. LQ-GNN for Resource Allocation and Elasticity

LQ-GNN is designed to predict the response time of microservice-based applications. By accurately forecasting how different deployment configurations influence the application's performance, LQ-GNN becomes a key tool for an efficient implementation of the resource allocation and elasticity mechanisms. An illustrative example of the use of LQ-GNN, which we are currently developing, would come

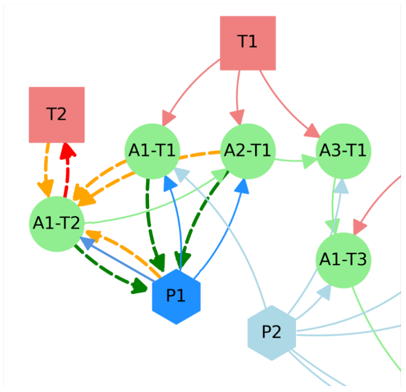

Fig. 4. Example graph with message-passing proposal.

from steering the resource allocation mechanism by concatenating a genetic algorithm (GA) and an optimization heuristic. The use of GAs is specially suitable to explore a wide search space of candidate deployment locations and resource assignments for any application. The goal is to meet the response time requirements while adhering to constraints such as cost or resource availability when several applications must be deployed simultaneously. The GA receives an LQN graph for a given application, the expected traffic demand, and the required response time. Starting with strategic candidate solutions, where a solution represents a specific deployment of microservices in the continuum (including how many instances of each microservice) and a given allocation of resources (e.g., CPU cores), the GA runs the LQ-GNN approach to estimate the response time of each proposed candidate solution. This way, the GA will be able to arrange an initial population of many different valid solutions, and in successive generations, the same GA will converge to a subset of optimal or near-optimal solutions, all meeting the required response time. This process is repeated for each of the applications to be deployed. Once optimal allocations are identified for each individual application, a higher-level heuristic will take those candidate solutions and select one of them for each application to coordinate the deployment of all applications together. This heuristic will balance the competing demands of different microservices and the available resources, ensuring the best possible placement throughout the entire infrastructure.

VI. Evaluation

In this section, we present the LQ-GNN assessment on predicting the response time of microservice-based applications. First, we provide a description of the methodology and data generation for training, as well as an overview of our experimental configuration. In the following, we conduct an evaluation of the performance of LQ-GNN and compare it against an LQN analytical solver.

A. Methodology

We evaluated the effectiveness of our proposed LQ-GNN model in predicting the response time of both individual microservices and the entire application. To achieve this, we tested the model on datasets generated from three different microservice-based applications: a 4-tier web application [20] (4-tier) and two variations of a common Social Network application [8] (SN-1 and SN-2). Furthermore, we conducted experiments using the Mix case, where the model was trained and tested with a combination of the three datasets to assess its generalizability.

We also considered an infrastructure made up of a set of computing servers distributed across the edge, fog, and cloud. In this setup, network latency varies depending on the distance between the end-user and the servers, as well as between servers located at different tiers of the continuum. Although our model can accommodate any network latency between pairs of microservices, we chose only three tiers for clarity and simplicity in the evaluation.

The datasets used for training and testing were obtained by implementing and executing the applications in an extended version of the μqSim Simulator [20]. μqSim provides comprehensive intra- and inter-microservice models, enabling accurate emulation of complex multi-tier applications. Our extended simulator[2] introduces additional capabilities, such as defining network delays between microservices (to model cloud-continuum placements), deploying multiple instances of microservices, and extracting detailed performance metrics for each microservice.

Each one of the three testing graphs contained a different number of microservices, connections, and paths. Even more, each microservice has its own set of activities, each of them with its processing delay and specific characteristics (e.g., blocking call behavior). Then, for each graph, a set of instances was generated by varying the following setup characteristics:

- number of threads assigned to each microservice (1 to 8),
- number of cores assigned to each microservice (1 to 8),
- placement of the microservices on edge, fog or cloud servers (with the corresponding variation on the network latency between microservices),
- request load (requests per second) of the application (200 to 20,000 depending on the graph).

For each graph, we conducted between 200K and 250K simulations by varying the features mentioned above. The total simulation time required to generate the dataset varies significantly by graph, ranging from 3,000 hours for the 4tier graph to 50,000 hours for the SN-1 graph. However, since the simulations are independent, they can be parallelized. In our case, we reduced the wall time for the SN-1 graph to just 300 hours through parallel execution.

The simulation results consisted of four different statistics of the application performance:

- Average end-to-end response time. This is the average response time of the application.
- 95th percentile end-to-end response time. This is the 95th percentile response time (i.e., tail response time) of the application.
- Average response time per microservice. This is the average response time of an individual microservice of the application.

[2]https://github.com/mrichart/uqsim-power-management-beta

- 95th percentile response time per microservice. This is the 95th percentile response time of an individual application microservice.

From all simulation results, we kept only those with a request rejection rate below 2%. This approach produced a total amount of approximately 100K samples for each graph, from which we took 80% for training, 10% for validation, and the remaining 10% for testing. All experiments were conducted on a server with an Intel Xeon Gold 6138 40-Core Processor CPU with a 40GB RAM running Linux CentOS 7.

In Tables I and II we present the main statistics for the response time from the training dataset. This will be helpful when evaluating the performance of predictions in the next sections.

TABLE I
APPLICATIONS RESPONSE TIME STATISTICS FROM THE TRAINING DATASET.

	Average response time (ms)					95th percentile response time (ms)				
Graph	Min	25th	Median	75th	Max	Min	25th	Median	75th	Max
4-tier	0.211	0.337	0.566	1.17	63.8	0.833	0.958	1.32	6.12	728
SN-1	1.09	1.96	3.40	5.66	121	3.60	8.95	14.0	21.1	930
SN-2	1.12	2.04	3.71	5.96	150	3.62	9.07	14.9	21.8	936
Mix	0.211	1.16	2.33	4.97	150	0.834	4.57	10.8	19.3	936

TABLE II
MICROSERVICES RESPONSE TIME STATISTICS FROM THE TRAINING DATASET.

	Average response time (ms)					95th percentile response time (ms)				
Graph	Min	25th	Median	75th	Max	Min	25th	Median	75th	Max
4-tier	0.00190	0.0131	0.450	4.51	490	0.00644	0.0347	1.293	14.5	2612
SN-1	0.0116	0.201	1.88	4.90	121	0.0297	0.602	10.0	15.15	930
SN-2	0.0117	0.166	1.65	4.70	150	0.0296	0.491	8.78	15.0	936
Mix	0.00190	0.200	1.55	4.80	490	0.00644	0.572	8.30	15.0	2612

B. LQ-GNN Implementation and Training

The proposed GNN model was implemented using the IGNNITION Framework [28], which allows a fast prototyping of the solution. The implemented model is available online[3].

The main implementation setup characteristics are:

- The dimension of all entities' hidden states is 8.
- The number of iterations (T) is set to 8.
- The neural network for generating the messages between activities is a two-layer, fully connected neural network with 32 and 8 units and ReLU activation.
- The readout neural network is implemented as a 3-layer fully connected neural network with ReLU activation function and 32 units for the hidden layers, and a linear one for the output layer.

For each of the three graphs, we train the GNN model, thus getting a different trained GNN for each graph. We use the training and validation dataset of the corresponding graph for training, while preserving the test dataset for evaluating the performance, using instances never seen during training.

[3]https://github.com/mrichart/lq-gnn

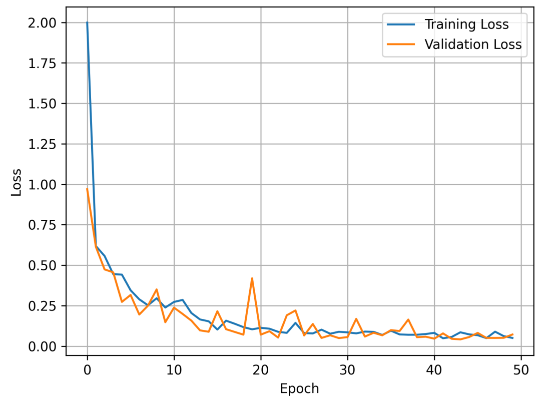

Fig. 5. Evolution of loss during training for the 4-tier graph.

In addition, we also trained a fourth GNN model using randomly selected instances from the three previous graphs. The rationale is to show the generalization capability of the approach; the trained GNN is not tied to a particular graph.

We set the Mean Squared Error as the loss function for training, and we use an Adam optimizer with an initial learning rate of 0.001. We train each model over 50 epochs. Each epoch has between 1,000 and 1,600 steps, depending on the size of the training dataset. For graphs 4-tier and SN-1 we set a batch size of 65, while for graph SN-2 and the Mix case we set the batch size to 128. All these values were empirically obtained so as to maximize prediction performance.

The average training time for the 4-tier graph was of 60 hours, for the SN-1 graph of 190 hours, for the SN-2 graph of 45 hours, and for the Mix case of 280 hours. In Figure 5, we show the evolution of the training and validation losses for the 4-tier graph. As we can see, it converges to a satisfactory low value in the 50 epochs used for training.

C. End-to-end Response Time

In this section, we show the performance of our proposed LQ-GNN model on predicting the end-to-end response time of the applications under evaluation.

We computed the error of the LQ-GNN predictions with respect to the results of the μqSim simulator using four different statistical metrics: Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), Mean Square Logarithmic Error (MSLE) and Root Mean Squared Error (RMSE). Our objective with MAPE and MSLE is to show the accuracy of our model. While MAPE is easily understood as a percentage, MSLE shows more stability when dealing with data with a wide range of values, such as the response times of individual microservices. On the other hand, MAE and RMSE show a measure of the prediction error in time units, which is very helpful to understand the expected performance of the proposed predictive model. While MAE is easier to interpret, we also include RMSE as it has a higher sensitivity to large error values.

In Tables III and IV we present the error results for predicting the average and the 95th percentile of the response time of the entire application compared to the results obtained with the simulator.

Regarding the average response time, the prediction accuracy of LQ-GNN is highly reliable when trained and tested on an individual application, achieving a MAPE between 5% and 7%. However, when using the generalized model (where the GNN is trained on a combination of the three graphs) the accuracy decreases, resulting in a MAPE of approximately 10%. For the other two statistical metrics, the MAE ranges between 0.2ms and 0.4ms, while the RMSE falls between 1.6ms and 2.1ms across all cases.

TABLE III
AVERAGE RESPONSE TIME PREDICTION PERFORMANCE OF LQ-GNN.

Graph	MAPE	MAE	MSLE	RMSE
4-tier	6.9%	0.239	0.0230	1.633
SN-1	4.6%	0.388	0.0118	2.191
SN-2	4.7%	0.383	0.0090	2.046
Mix	9.7%	0.342	0.0459	2.102

To better understand the significance of the obtained accuracy values, it is important to consider the objective of these predictions. As described in the introductory section, the goal of our LQ-GNN predictions is to assist an elasticity controller in making informed decisions regarding the placement and resource allocation of an application. Given that response times in microservice-based applications typically range from several milliseconds to hundreds of milliseconds, a mean absolute error of 0.4 ms represents only a small fraction of the total response time. This level of error is well within an acceptable range for practical decision-making, as it does not significantly impact the controller's ability to optimize resource allocation effectively.

For the tail response time (95th percentile), the MAPE is slightly higher, ranging from 7% to 10%, with an MAE of around 2ms. However, considering that the response time values in this case are significantly higher and more varied (see Table I), these results are still very promising. Once again, for the purpose of considering such predictions, the errors obtained are satisfactory.

TABLE IV
95TH PERCENTILE RESPONSE TIME PREDICTION PERFORMANCE OF LQ-GNN.

Graph	MAPE	MAE	MSLE	RMSE
4-tier	7.4%	1.747	0.0210	12.533
SN-1	6.9%	2.539	0.0374	15.050
SN-2	7.3%	2.171	0.0213	13.838
Mix	9.6%	2.081	0.0290	15.081

D. Microservice Response Time

We also evaluated the performance of LQ-GNN on predicting the response time of individual microservices of the application. Having an individual prediction for each constituent microservice would allow an elasticity controller to have a more detailed diagnosis of the performance of the application. In particular, this would help finding microservices which are generating high response times and decide, for example, to increase their resources or change their placement.

As in the previous section, we report the error of LQ-GNN prediction with respect to the results of the μqSim simulator using the same error metrics. In Tables V and VI, the average response time and the results of the 95th percentile response time are presented. To ease visualization, we calculated the error for each microservice of the application and averaged by the total number of microservices.

TABLE V
INDIVIDUAL MICROSERVICE AVERAGE RESPONSE TIME PREDICTION PERFORMANCE OF LQ-GNN.

Graph	MAPE	MAE	MSLE	RMSE
4-tier	4.7%	0.422	0.0126	5.344
SN-1	3.2%	0.185	0.0079	1.459
SN-2	6.5%	0.301	0.0199	1.569
Mix	9.5%	0.438	0.0114	5.546

Similarly to the case of the end-to-end response time, the results are highly accurate. For the average response time of the three graphs, the MAPE ranges between 3% and 6.5%, with an MAE between 0.2ms and 0.4ms. For the case of the 4-tier graph, we can observe a higher RMSE compared to the others. This is because, in this scenario, the variability of the response time among the different microservices is higher (see Table II) and includes a considerable amount of outliers. As in previous results, the more general model has a higher error with a MAPE of 9.5%, but still maintaining a good prediction accuracy in the other metrics.

TABLE VI
INDIVIDUAL MICROSERVICE 95TH PERCENTILE RESPONSE TIME PREDICTION PERFORMANCE OF LQ-GNN.

Graph	MAPE	MAE	MSLE	RMSE
4-tier	4.5%	1.492	0.0122	21.011
SN-1	5.6%	1.313	0.0215	9.825
SN-2	7.4%	1.493	0.0271	9.800
Mix	7.3%	2.281	0.0329	27.553

For the prediction of tail response time (95th percentile) of a microservice, we have very similar performance results. In this case, we obtain a MAPE between 4.5% and 7.5% and a MAE of around 1.5ms. Again, given the broader distribution of different response times in the dataset of the 4-tier graph, the RMSE for this case (and for the Mix case) is much higher.

E. LQN Analytical Solver

To compare our results against other potential prediction approaches, we have used an LQN analytical solver. The LQN solver [13] consists of an iterative algorithm based on the Linearizer algorithm [29]. The Linearizer is a heuristic algorithm to obtain approximate average statistics from basic queueing-network. Briefly explained, the LQN solver works by applying the Linearizer algorithm to each layer of the LQN graph in an iterative manner until a convergence threshold is reached. Therefore, it is important to note that the LQN solver does not provide an exact solution to the problem because of its coarse-grain approach.

For the LQN solver, we have used the same datasets and performed the same tests for each graph. As we did with LQ-GNN, we compare the response times found by the solver with the results obtained by the μqSim Simulator. The results obtained were not very promising, with an MAPE between 50% and 60%, depending on the graph, for both the end-to-end response time and the microservices response time. Therefore, we discarded providing a more detailed comparison.

Moreover, to validate the model implemented for the LQN solver (note that the model needs to be described in a specific *XML Schema* [4]), we tested the same specified LQN model by an LQN simulator [16]. For the LQN simulator, it was not possible to run the entire test dataset since each simulation takes between 60 and 120 minutes to complete and it would have been necessary to carry out up to 250K simulations. However, we randomly selected 200 instances from the 4-tier graph and ran them in the LQN simulator. Again, we compare the results of the LQN Simulator with those of the μqSim Simulator. We obtained an average MAPE of 15%, suggesting that the model implemented is accurate and that is the LQN solver that does not produce good results for our test cases.

VII. Conclusions and Future Work

In this work, we have analyzed the challenges and solutions involving the placement and resource allocation of microservice-based applications in a computing continuum working scenario. By focusing on providing elasticity for the deployment of applications onto a computing continuum, we emphasize the requirement to have a performance modeling tool of microservice-based applications.

In this regard, we propose LQ-GNN, a novel strategy to predict the response time of microservice-based applications while considering different microservice locations and resource assignments. We demonstrated that the LQ-GNN model, leveraging Layered Queueing Networks and Graph Neural Networks, provides a robust framework for predicting applications performance. This model offers significant improvements over existing approaches, making it a feasible tool for real-time resource allocation and elasticity control. The results showed that the model assures acceptable error margins even when generalized across multiple applications, ensuring its practical applicability in diverse scenarios.

Moreover, we have highlighted the limitations of current approaches, such as the computational cost of simulations, the unrealistic assumptions of analytical solvers, the generalization shortage of similar ML solutions, and how our approach addresses those shortcomings. In summary, our work contributes to providing a valuable tool in the decision-making of resource allocation approaches allowing for more accurate and timely solutions.

Future research will further refine this model, exploring its application in increasingly complex and heterogeneous cloud computing environments as well as for more intricate microservice-based applications. We also plan to extend our testing with real-world data, aiming to validate and refine our model's performance in practical environments. This step will allow us to better understand the challenges and opportunities presented by real-world applications and improve the robustness and versatility of the model.

[4]http://www.sce.carleton.ca/rads/lqns/lqn-documentation/schema/

Acknowledgments

This work has been partially funded by the project "UNICO-5G I+D-OPTIMAIX-TSI-063000-2021-34", by the Beatriu de Pinós Programme of the Ministry of Research and Universities of the Government of Catalonia under grant 2022-BP-00099 and also supported by the Agencia Estatal de Investigación of Ministerio de Ciencia e Innovación of Spain under project PID2022-137329OB-C41 /MCIN/AEI/10.13039/501100011033.

References

[1] R. M. Swoyer, Steve, "Cloud Adoption in 2020," May 2020. [Online]. Available: https://www.oreilly.com/radar/cloud-adoption-in-2020/

[2] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, Fog, or Edge: Where to Compute?" *IEEE Internet Computing*, vol. 25, no. 4, pp. 30–36, Jul. 2021, conference Name: IEEE Internet Computing.

[3] M. L. Swoyer, Steve, "Microservices Adoption in 2020," Jul. 2020. [Online]. Available: https://www.oreilly.com/radar/microservices-adoption-in-2020/

[4] O. Zimmermann, "Microservices tenets," *Computer Science - Research and Development*, vol. 32, no. 3, pp. 301–310, Jul. 2017. [Online]. Available: https://doi.org/10.1007/s00450-016-0337-0

[5] J. Baliosian, L. M. Contreras, P. Martínez-Julia, and J. Serrat, "An Efficient Algorithm for Fast Service Edge Selection in Cloud-Based Telco Networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 34–40, Oct. 2021, conference Name: IEEE Communications Magazine. [Online]. Available: https://ieeexplore.ieee.org/document/9627830

[6] J. Chapel, "Wasted Cloud Spend to Exceed $17.6 Billion in 2020, Fueled by Cloud Computing Growth," Mar. 2020. [Online]. Available: https://jaychapel.medium.com/wasted-cloud-spend-to-exceed-17-6-billion-in-2020-fueled-by-cloud-computing-growth-7c8f81d5c616

[7] M. H. Fourati, S. Marzouk, and M. Jmaiel, "Cloud Elasticity of Microservices-based Applications: A Survey," In Review, preprint, Feb. 2024. [Online]. Available: https://www.researchsquare.com/article/rs-3925329/v1

[8] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. Providence RI USA: ACM, Apr. 2019, pp. 3–18. [Online]. Available: https://dl.acm.org/doi/10.1145/3297858.3304013

[9] J. Park, B. Choi, C. Lee, and D. Han, "GRAF: a graph neural network based proactive resource allocation framework for SLO-oriented microservices," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. Virtual Event Germany: ACM, Dec. 2021, pp. 154–167. [Online]. Available: https://dl.acm.org/doi/10.1145/3485983.3494866

[10] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: ML-based and QoS-aware resource management for cloud microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Virtual USA: ACM, Apr. 2021, pp. 167–181. [Online]. Available: https://dl.acm.org/doi/10.1145/3445814.3446693

[11] D. S. H. Tam, Y. Liu, H. Xu, S. Xie, and W. C. Lau, "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*. Long Beach CA USA: ACM, Aug. 2023, pp. 2155–2165. [Online]. Available: https://dl.acm.org/doi/10.1145/3580305.3599465

[12] H. X. Nguyen, S. Zhu, and M. Liu, "Graph-PHPA: Graph-based Proactive Horizontal Pod Autoscaling for Microservices using LSTM-GNN," Sep. 2022, arXiv:2209.02551 [cs]. [Online]. Available: http://arxiv.org/abs/2209.02551

[13] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi, "Enhanced Modeling and Solution of Layered Queueing Networks," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 148–161, Mar. 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4620121/

[14] A. Amazon, "Application Scaling - AWS Auto Scaling - AWS," 2024. [Online]. Available: https://aws.amazon.com/autoscaling/?nc1=h_ls

[15] Kubernetes, "Horizontal Pod Autoscaling," 2024, section: docs. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[16] G. Franks, P. Maly, M. Woodside, D. C. Petriu, M. Mroz, and A. Hubbard, "Layered Queueing Network Solver and Simulator User Manual," Department of Systems and Computer Engineering, Carleton University, Tech. Rep., 2022.

[17] A. U. Gias, G. Casale, and M. Woodside, "ATOM: Model-Driven Autoscaling for Microservices," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. Dallas, TX, USA: IEEE, Jul. 2019, pp. 1994–2004. [Online]. Available: https://ieeexplore.ieee.org/document/8884900/

[18] E. Incerto, R. Pizziol, and M. Tribastone, "μOpt: An Efficient Optimal Autoscaler for Microservice Applications," in *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, Sep. 2023, pp. 67–76. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10336041

[19] A. Arfeen, K. Pawlikowski, D. McNickle, and A. Willig, "The role of the Weibull distribution in modelling traffic in Internet access and backbone core networks," *Journal of Network and Computer Applications*, vol. 141, pp. 1–22, Sep. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804519301547

[20] Y. Zhang, Y. Gan, and C. Delimitrou, "uqSim: Scalable and Validated Simulation of Cloud Microservices," Nov. 2019, arXiv:1911.02122 [cs]. [Online]. Available: http://arxiv.org/abs/1911.02122

[21] H. X. Nguyen, S. Zhu, and M. Liu, "A Survey on Graph Neural Networks for Microservice-Based Cloud Applications," *Sensors*, vol. 22, no. 23, p. 9492, Jan. 2022, number: 23 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/22/23/9492

[22] G. P. Gu and D. C. Petriu, "Xslt transformation from uml models to lqn performance models," in *Proceedings of the 3rd international workshop on Software and performance*, 2002, pp. 227–234.

[23] ——, "From uml to lqn by xml algebra-based model transformations," in *Proceedings of the 5th international workshop on Software and performance*, 2005, pp. 99–110.

[24] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice Architecture Reconstruction and Visualization Techniques: A Review," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Aug. 2022, pp. 39–48, iSSN: 2642-6587. [Online]. Available: https://ieeexplore.ieee.org/document/9912633/?arnumber=9912633

[25] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating service resource consumption from response time measurements," in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS '09. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Oct. 2009, pp. 1–10. [Online]. Available: https://dl.acm.org/doi/10.4108/ICST.VALUETOOLS2009.7526

[26] M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, K. Rusek, S. Xiao, X. Shi, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet-Fermi: Network Modeling with Graph Neural Networks," *IEEE/ACM Transactions on Networking*, pp. 1–0, 2023, arXiv:2212.12070 [cs]. [Online]. Available: http://arxiv.org/abs/2212.12070

[27] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[28] D. Pujol-Perich, J. Suárez-Varela, M. Ferriol, S. Xiao, B. Wu, A. Cabellos-Aparicio, and P. Barlet-Ros, "Ignnition: Bridging the gap between graph neural networks and networking systems," *IEEE Network*, vol. 35, no. 6, pp. 171–177, 2021.

[29] K. M. Chandy and D. Neuse, "Linearizer: a heuristic algorithm for queueing network models of computing systems," *Communications of the ACM*, vol. 25, no. 2, pp. 126–134, Feb. 1982. [Online]. Available: https://dl.acm.org/doi/10.1145/358396.358403

VIII. Biography Section

Matías Richart is currently an Assistant Professor at University of the Republic (UdelaR) in Uruguay and a visiting researcher at Polytechnic University of Catalonia (UPC) in Spain. He received his Computer Engineer and Master degree from UdelaR in 2011 and 2014 respectively and his Ph.D. from UdelaR and from UPC in 2019. His research focuses on the autonomous control and management of networks and services through optimization and machine learning techniques. He also has vast experience in network simulations.

Juan-Luis Gorricho received a network engineering degree and Ph.D. degree from the Technical University of Catalonia (UPC) in 1993 and 1998, respectively. Since 2001 he is Associate Professor at the Network Engineering department of the UPC. His more recent research interests include the development of optimization and artificial intelligence techniques for the management of network and services in edge-fog-cloud computing environments.

Javier Baliosian specializes in computer science, particularly in the areas of computer networks, network management, and autonomous systems. He earned his bachelor's degree in computer engineering from the University of the Republic in Uruguay in 1998 and his PhD from the Polytechnic University of Catalonia in Spain in 2005. Currently, Baliosian is a Full Professor at the University of the Republic, where he continues his research, focusing on solutions for network management and optimization.

Luis M. Contreras holds an M.Sc. in Telecommunications, an M. Sc. in Telematics, and a Ph.D. in Telematics. Since August 2011 he is part of Telefónica, working on scalable networks and their interaction with cloud and distributed services, and participating in several international projects (currently under the programmes Horizon Europe and Smart Network and Services). He is an active contributor to different SDOs, such as IETF, O-RAN, ETSI, etc

Alejandro Muñiz received his Computer Engineering degree from Universitat Politècnica de València (UPV) and a Master's in Cybersecurity from Universidad Politécnica of Madrid (UPM). He works at Telefónica Innovación Digital, focusing on IP network protocols for transport networks. He participates in European and national research projects and is pursuing a Ph.D. in Communications Technologies and Systems at UPM.

Joan Serrat-Fernández received the degree of Telecommunication Engineer in 1977, and the Doctor degree in Telecommunication Engineering in 1983, both from Universitat Politècnica de Catalunya (UPC) in Barcelona, Spain. Since 2022 he is an Emeritus Professor at UPC, involved in projects of research and promotion of Telecommunication studies. His topics of expertise are in the field of autonomic networking and service and network management.