IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 July 2024, accepted 23 July 2024, date of publication 29 July 2024, date of current version 6 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3434585

RESEARCH ARTICLE

Adapting Containerized Workloads for the Continuum Computing

ALBERTO ROBLES-ENCISO AND ANTONIO F. SKARMETA, (Senior Member, IEEE)
Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain

Corresponding author: Alberto Robles-Enciso (alberto.roblese@um.es)

This work was supported in part by Seneca Foundation in Region of Murcia (Spain) under Grant 21463/FPI/20, and in part by European Union's Horizon Europe Research and Innovation Program through the FLUIDOS Project under Grant 101070473 and through RIGOUROUS under Grant 101095933.

ABSTRACT Container and microservices management platforms are currently one of the most important tools for cloud computing, but since the scope of these tools is homogeneous cloud architectures they have serious limitations in adapting to new computing paradigms. Therefore, using the default scheduler in heterogeneous node systems faces significant limitations when tasked with orchestrating workloads in a Continuum Computing environment, as the nodes have very different characteristics and restrictions. To solve this limitation we decided to use Kubernetes as it is the most popular Container management tool and we propose to replace the native scheduler with a reimplementation that gives us complete flexibility for the process of assigning pods to nodes, providing a framework to design algorithms that considers all the necessary parameters for the deployment of services in a Continuum. In addition, we address one of the most limiting aspects of the K8s scheduler, its pod-by-pod allocation approach, which makes it difficult to optimise the complete set of allocations. To test our proposal we design a use case and perform several tests on a real environment based on virtual machines, in which stress tests are conducted to measure the performance of each method. We then present a series of results to justify the benefits of our proposal, including the reduced performance provided by the pod-pod approach and how a batch-based approach greatly improves efficiency. The results show the usefulness of using batch-based approaches and how the Kubernetes scheduler extension points are not enough to support the requirements of the Continuum.

INDEX TERMS Scheduling, containers, resource allocation, computing continuum, edge computing.

I. INTRODUCTION

As the demand for scalable and efficient computing solutions continues to grow, cloud-based solutions were initially the best approach to address this need. However, the surge in online services and users has made it challenging to sustain the cloud computing paradigm due to increasingly congested cluster bandwidth and the rising demand for ultra-low latency services.

From these requirements emerges the promising Edge Computing paradigm which proposes to delegate as much as possible of the computation to specially integrated nodes in areas close to the users, thus reducing latency considerably and limiting bandwidth usage as they process locally the largest amount of data without the need to communicate with the Cloud node. To further increase the performance, another layer of nodes called Fog is usually integrated, which has more computational power (slightly similar to the Cloud level) but is closer to the Edge nodes, so you have the popular Edge-Fog-Cloud architecture. Even so, it remains a passive model where decisions are made solely about whether to send data to one level or another. Alternatively, users can only interact with the Edge layer, which aggregates information to progressively reduce its volume (while increasing its semantics) before it reaches the Cloud level.

In this context, the Computing Continuum appears as an innovation to take advantage of the Edge-Fog-Cloud architecture by conceptually merging all layers into a single computing continuum, allowing developers the flexibility to implement their applications in a distributed manner

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

across an ecosystem of various types of nodes located in distant locations. Since it is necessary to operate with the complete set of nodes, it is important to consider their individual characteristics, as it is a heterogeneous and diverse set of devices. This includes some nodes with very low computational capacities and others with high computational power, including specialized acceleration units such as GPUs. Therefore, it is crucial to optimally allocate services (tasks) to the nodes that are best suited according to various criteria. This challenge is known as the Task Allocation Problem (TAP), which involves determining the most efficient way to distribute computational tasks across the available nodes to maximise performance and ensure efficient resource utilisation.

On the other hand, container orchestration solutions address the needs of cloud computing and microservices, but a significant proportion of container management systems do not explore new paradigms and focus on ensuring their operation in Cloud Computing architectures, being hardly adaptable to more complex networks such as Continuum Computing. One of the most famous tools in the service orchestration domain with the potential to adapt to Continuum is the well-known platform Kubernetes (K8s), an open-source platform born out of Google's experience with managing containerized applications at scale, Kubernetes has become de facto standard for automating deployment, scaling, and management of containerized applications. Its architecture, built around a cluster of nodes, provides a platform-agnostic solution for container orchestration, enabling seamless scaling and resilience across diverse environments. Thus, K8s is a very useful tool for the computing continuum but it has some very important limitations that reduce the effectiveness of its implementation due to its fundamental emphasis on cloud computing.

A. MOTIVATION AND CONTRIBUTIONS

As already mentioned, the de facto tool for deploying and managing containers (applications and services) is Kubernetes, so we will focus on adapting it, although our solution can easily be applied to other similar tools. However it is not possible to use the standard version directly to handle a computing continuum, hence this work aims to describe the limitations of container orchestration systems and to propose solutions to adapt its features to the continuum.

While Kubernetes has revolutionized container orchestration, its default scheduler faces significant limitations when tasked with orchestrating workloads in continuum computing environments. In particular, the conventional scheduler approach to assigning pods to nodes is inadequate because it neither considers the characteristics of the nodes nor the conditions of the network. In addition, K8s offers several extension points to modify the operation of the scheduler, but these are still very basic and do not allow for the deployment of a sophisticated decision engine.

In this context, we consider that it is appropriate to reimplement the scheduler to become an independent module of the K8s core in order to be enabled to perform more complex scheduling decisions, by acting as a middle point between application deployments by users and the K8s cluster. Furthermore, this module will be able to make pod-node allocation decisions on a complete set of services (e.g. a deploy) achieving a more optimal allocation, contrary to the greedy one-by-one pod allocation approach of Kubernetes vanilla.

Therefore, the main contributions of this work are the following:

- We defined a Computing Continuum architecture using Kubernetes which includes different layers of devices (edge-fog-cloud model), each one with different capabilities and computing power.
- We have implemented a module that manages K8s deployment requests (yaml files) and then sends them to Kubernetes including the required metadata for pod-node allocation.
- We have also reimplemented the native K8s scheduler to include the additional metadata that has been introduced, and also to be able to communicate directly with our orchestration module to resolve the allocation.
- We have implemented a greedy algorithm for the individual pod-node allocation following a similar approach to Kubernetes but including the necessary constraints in Computing Continuum (computational power, latency, bandwidth, capabilities, location, etc.).
- In addition, we have designed an optimal algorithm (based on backtracking) to solve the problem of allocating a set of services (several pods at once) to the nodes, achieving a more optimal allocation concerning the greedy algorithm.
- To test the methods we have designed a real test scenario with virtualised Kubernetes nodes, modified to have the real performance that we defined in the tests.
- In addition, we have also implemented a docker image to simulate the CPU usage when receiving a service request over time according to the computational characteristics of the node and the requirements of the service. In this way, we can simulate the realistic behaviour of different types of applications and perform stress tests on the infrastructure.
- Finally, the result of our proposal is analysed to demonstrate how the basic Kubernetes scheduler operation is not suitable to consider the requirements of the Computing Continuum.

B. ORGANIZATION

The rest of the paper is organized as follows. In section II we explore the state of the art of the continuum and the allocation problem. In section III we introduce the task assignment problem with its main components. In section IV we present our proposals to adapt the Kubernetes scheduler to the Computing Continuum paradigm. Then, in section V we evaluate the proposed solution in different scenarios and analyse the results. Finally, the conclusions and future work are drawn in section VI.

II. STATE OF THE ART

The process of deciding what resources to maximise the efficiency of an infrastructure is a well-known problem, however with the emergence of the concept of Continuum Computing new ideas and challenges emerge in terms of determining how to manage this new type of network in the most efficient way [1]. To verify the feasibility, the possibilities of Continuum are explored with experiments and simulations to discover the possible future potential of this kind of distributed infrastructure, as for example Daniel Rosendo et al. describe in [2]. However, the management of the Continuum requires considering its heterogeneous characteristics, something that is not common in the methods applied in Cloud Computing. In [3] Marchese and Tomarchio present in their paper how it is necessary to modify standard task schedulers to consider new features that are very relevant in the process of deciding the best node to perform a job, such as bandwidth usage or the cost of using it. Similarly, Fu et al. exploit the Continuum to deploy microservices and for this purpose they developed in [4] a system (Nautilus) in charge of deploying the services in an optimal way, ensuring QoS and always considering the use of the network, since it is an important factor in the computing continuum due to its direct impact on the performance of the services.

Although the concept of Continuum Computing is relatively new, it benefits from the fact that the problem of managing computing resources efficiently in a network of nodes is a research area that has been extensively studied by Cloud Computing and recently significantly improved by Edge Computing. Therefore, there are a large number of works proposing all kinds of algorithms, methodologies and frameworks for task and node management [5], [6], being of special interest the new approaches based on Edge Computing [7], which are easily adaptable to the Continuum Computing paradigm. As mentioned, the continuum differs from Cloud Computing by having to address new constraints in the process of deciding nodes, for example, it is important to take into account energy consumption [8], the geolocation of the node [9], latencies [10], and even if the node is moving [11]. To achieve this, various algorithms are employed. However, due to the typically high computational complexity of the problem, alternative techniques are often utilized, with a recent focus on those based on evolutionary methods [12]. Additionally, even novel techniques such as Reinforcement Learning have been enhanced [13] to achieve better integration into Edge-Fog-Cloud systems, allowing to dynamically learn in which layer the offloading of a task is most appropriate.

In multi-node systems where services are deployed, Kubernetes stands as the de facto standard. This container orchestrator has gained immense popularity in the cloud computing and microservices management industry due to its robust capabilities. However, the default performance of K8s is limited in architectures other than relatively homogeneous node clusters and is therefore not typical for use in heterogeneous distributed systems. Even so, these limitations are known and discussed by several works [14], [15], which provide methodologies for improvement supported by the tools that Kubernetes offers to extend the functionality of its components, for example the scheduling Framework. However, these works tend to focus mainly on Cloud computing architectures, as for example in the paper by Lebesbye et al. [16] where the limited performance of the K8s default scheduler is identified and they propose a service (Boreas) which provides a better way to select the best nodes for each of the services but within a single Cloud cluster. Similarly, Marchese and Tomarchio [17] propose to improve the K8s scheduler for its limited performance by taking into account network-related parameters, extending the architecture to an edge computing environment where it is critical to consider the quality of the connection between nodes. In addition, other works also address this issue, such as [18] where the concept of Computing Continuum is also introduced, while others also report an increasing interest in adding new features to the scheduling, such as the geolocation of nodes [19], which are very important for the continuum, thus serving as a reference for future research. The paper by Sebastian Böhm and Wirtz [20] discusses in a general way the difficulties of orchestrating Cloud-Edge architectures in Kubernetes and proposes solutions, being one of the few to explicitly discuss many of the critical limitations of the K8s scheduler, for example, one of the most notable is the one-to-one mapping approach of the pod to node allocation process.

As a conclusion, there are many works dealing with node task allocation, intelligent container management and similar, so this work does not aim to propose different methods for optimal resource management. The main objective is what has been detected in the literature review, there is a limited number of papers and tools that focus on improving the most important orchestrator of Cloud Computing to be efficient in the new computing paradigms. Although the area of Edge Computing is being addressed in recent works, the new Continuum approach that combines the classic Cloud model with Edge and Fog is not yet a very advanced area of study, especially in the necessity to incorporate the differential characteristics of each node (compute power, hardware accelerators, specific capabilities, etc.) into the orchestrator resource management process. Therefore, we consider relevant to try to integrate the individual properties of the nodes in the resource management, in particular the aspects related the allocation of workloads to nodes. Consequently, when the orchestrator has to decide to which node to send a job, it can consider the capabilities and performance of the node, thus being able to select the node that for example solves the problem faster (higher computational power) although it has more latency, or the node that is from a specific country and has a hardware accelerator (GPU) available. In short, the orchestrator will consider the heterogeneous nature of its nodes when choosing where to send computing tasks, so as to optimise the use of the infrastructure and improve QoS.

III. TASK ASSIGNMENT PROBLEM

The main problem that orchestrators of applications have to handle in a computing continuum is to determine in which parts of the infrastructure it is most convenient, according to all kinds of criteria, to deploy the services in order to optimise the overall use of resources. This problem is called the Task Assignment Problem (TAP) which is considered a combinatorial optimisation problem commonly referred to as the Assignment Problem [21].

In general terms, the assignment problem is defined as: Given a set of tasks and a set of agents, each task can be assigned to any of the agents incurring a cost and providing a benefit, as long as the agent's capacities are not exceeded. The objective is to find the best task-agent assignment of all tasks that maximises the final benefit without exceeding the capabilities of each agent. The capabilities of an agent can be, in addition to the typical CPU and RAM, others such as its location, CPU speed, average latency, available bandwidth, GPU or RAM-ECC availability.

In this context, orchestrators ideally aim to solve the TAP for each set of applications they want to deploy, but since solving this problem is usually costly, they try to use another methodology such as one-to-one assignments based on heuristics. In the case of the continuum computing paradigm, this problem is further complicated by the heterogeneous nature of the devices that are part of the infrastructure (from mobile phones to large datacenters), the widespread physical locations between nodes, the limited bandwidth at some nodes, and many other details. However, the different capabilities of the elements of this sort of architecture can be exploited to distribute services and applications over the continuum according to their priority and computational requirements, thus minimising the cost of the applications while maximising their performance.

A. ELEMENTS

As mentioned above, the elements that constitute the infrastructure of the computing continuum are very different in terms of both capacity and design. In the areas closest to the user we have smaller devices with limited capacities but capable of responding quickly to requests (low latency), at the other side we have Cloud servers with large computing capacity but with higher latency and less available bandwidth. In between the two sides, different devices with intermediate capabilities are deployed to support the needs of users and developers. Figure 1 summarises the typical distribution of devices in a Continuum by identifying the services that could be deployed at each point according to latency, bandwidth and compute capacity.

Moreover, depending on the layer some devices may provide additional computing capabilities. For example, some Fog nodes may have GPUs to accelerate deep learning tasks while some Edge nodes may not have certain instructions needed to speed up encryption tasks (e.g. AES-NI). It is also important to note that the computational speed of each device is different, so the same tasks can be solved faster on a more powerful node even if it has higher latency. This variety of characteristics must be carefully considered by the orchestrator when deciding which node to assign a service to.

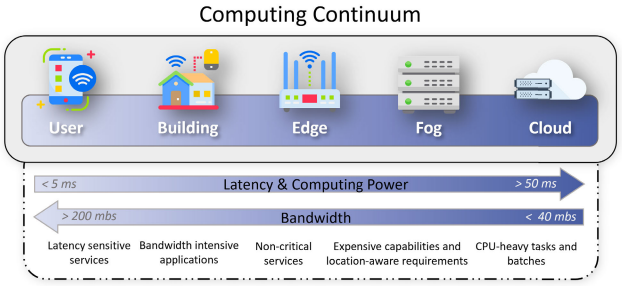

FIGURE 1. Computing continuum.

B. FORMAL DEFINITION

The task allocation problem is a well-known problem and can be formalised as a Generalized Assignment Problem (GAP) [22], which is an NP-hard [23]. The objective of the problem is to define the best allocation of tasks to the nodes, therefore the output of the algorithm will consist of an array of size S (number of services to be deployed), where each element will denote on which node (of the set N) that service is deployed. Hence, the output solution to the problem is defined as below:

$$\begin{aligned} &tw = \{tw_0, tw_1, \ldots, tw_{n-1}, tw_s\} \\ &\text{where:} \\ &tw_x = \{w \mid w \in N\} \end{aligned} \tag{1}$$

Therefore, the optimisation problem of assigning S services to N nodes is formulated as follows:

$$\begin{aligned} &\min \sum_{w=0}^{n} \sum_{t=0}^{s} a(w,t)\, c(w,t) \\ &\text{subject to: } \sum_{t=0}^{s} a(w,t) S^t_{cpu} \leqslant N^w_{cpu} \quad \forall w \\ &\sum_{t=0}^{s} a(w,t) S^t_{ram} \leqslant N^w_{ram} \quad \forall w \\ &\sum_{w=0}^{n} a(w,t) = 1 \quad \forall t \\ &\sum_{w=0}^{n} a(w,t) \sum_{t'=0}^{s} a(w,t') \cdot \mathrm{I}(t' \in t_{\text{anti}}) = 0 \quad \forall t \\ &a(w,t) = 0 \quad \textit{if } S^t_{loc} \neq N^w_{loc} \\ &a(w,t) = 0 \quad \textit{if } S^t_{reqCap} \subset N^w_{cap} \\ &a(w,t) = 0 \quad \textit{if } S^t_{maxLat} > N^w_{lat} \\ &a(w,t) \in \{0,1\} \end{aligned} \tag{2}$$

where the function $a(w,t)$ denotes whether a service t has been assigned at a node w, such that:

$$a(w,t) = \begin{cases} 1 & \text{if } tw_t \textit{ [eq 1]} \text{ is equal to } w \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

And the function $c(w, t)$ denotes the cost of assigning the node w to service t, which is expressed as the following weighted sum of different parameters:

$$c(w,t) = \alpha_1 w_{networkDelay} + \alpha_2 w_{cpuSpeed} + \alpha_3 (t_{desiredLatency} - w_{latency}) \quad (4)$$

The constraints of the problem described in eq. 2 represent respectively: Node assignments cannot exceed the CPU capacity of each node; Similarly, the RAM capacity of nodes can never be exceeded; No more than one node can be assigned per pod; Nodes will not have any services that have anti-affinity between them; A task is assigned to a node only if it has the same location, capabilities and required latency.

C. KUBERNETES LIMITATIONS

In this subsection we will present K8s specific limitations to solve the TAP in an optimal way. As we said before, although our work is focused on Kubernetes, our proposal can be applied to other container platforms and service systems as they usually share the same limitations. By default, Kubernetes provides a scheduler that has a very basic behaviour, and while it works well in general, it has limited flexibility when more complex requirements are needed. Therefore, K8s offers a whole framework to enhance/extend it according to the particular needs of each cluster by making use of a series of extension points to develop plugins. Figure 2 (from [24]) summarises the extension points they offer in each part of the scheduling process.

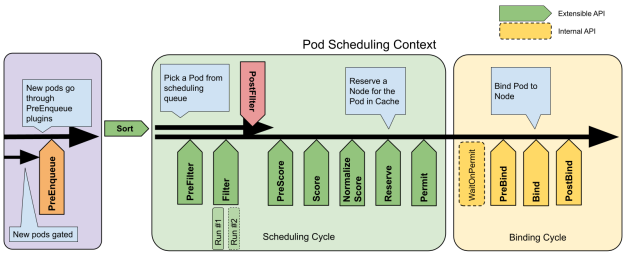

FIGURE 2. Kubernetes Scheduling Framework, extracted from [24].

Through the use of extension points (e.g. Filter and Score) the scheduler can be enhanced to consider the basic features of the Computing Continuum, e.g. giving better scores (using Score) to nodes that are geographically closer and for latency, as well as excluding nodes (using Filter) that are not valid due to their location or capabilities.

However, all these improvements have the limitation of being bound to the predefined flow of steps of the K8s scheduler, making it difficult to implement more complex methodologies that require performing several different steps or repeating them. Moreover, their proposed scheme is inherently limited in that it offers sub-optimal solutions that are likely to be far from the best solution to the problem.

When Kubernetes receives a request to deploy a service (denominated Pod) it enqueues it in a list of Pods to be assigned to a node. Periodically that queue is queried and pods are sent one by one to the scheduler to decide on which node will be allocated, resulting in a greedy pod-pod allocation. This approach therefore limits the possibility to optimise the allocation of a complete set of services by allocating them individually and separately. To group pods it is necessary to completely reimplement the scheduler so that it is able to select more than one pod at a time from the queue of pods to be assigned. In this paper we have adopted this approach, implementing a new scheduler fully customised to include all the needs of the Computing Continuum architecture. Nevertheless, it is important to note that since the optimisation problem is computationally very expensive, alternative solutions must be implemented to solve problems that are not of small size, e.g. genetic algorithms.

On the other hand, by reimplementing the scheduler it is also possible to apply more complex techniques when solving the task assignment problem, even considering the pod-by-pod approach of K8s. For example, if it is not important to optimise the complete set of services, pod-to-pod allocations can be kept to solve a multi-objective problem (e.g. latency and consumption) by using the best point of a Pareto front.

Therefore, the benefits of designing our own scheduler are the improvement in both flexibility and functionality of the default Kubernetes module, including the possibility of being able to solve the deployment of a set of services (pods) at once to obtain the optimal solution to the allocation problem, instead of the greedy sub-optimal pod-to-pod approach of K8s. Moreover, any other parameters can be easily considered in the node selection process, such as choosing the best node based on processor computing power, distance, latency, available bandwidth, or even excluding nodes that do not meet capacity requirements or specific characteristics (e.g. GPU). Along with the constraints that can be defined (e.g., maximum latencies, anti-affinity, location), soft-constraints can also be defined as desirable but not mandatory objectives in the optimisation process (e.g., desired average latency). In addition, as mentioned above, the pod-to-pod deployment approach can be maintained but the process can be enriched with more sophisticated methods such as using the Pareto front for multi-objective optimisation, more advanced heuristic algorithms and even neural networks.

IV. PROPOSED APPROACHES

Given that we have replaced the Kubernetes scheduler with our custom module that is responsible for offloading node-pod allocation requests to another external module, it is necessary to implement several allocation methods since K8s would be completely untethered from that role. Since our work is contextualised in the Continuum Computing paradigm, all the proposed methods will take into account the characteristics and requirements of the services that are deployed in this type of infrastructure, thus addressing the limitations of K8s.

In our proposals, two main categories can be defined, the methods that make a pod-node allocation one by one (K8s style) and the methods that make allocations from

several pods to several at a time (batch) to achieve a better optimisation of resources. The purpose of pod-pod methods is to provide a quick solution for problems where the global optimisation of the solution is not so important, but always considering the constraints of the Continuum. On the other hand, batch allocation solutions aim to optimise the allocation of a given set of pods (5, 10, etc…), always considering that the optimisation of the whole set of pods is difficult to achieve since either there are many pods and the optimisation problem is computationally very complex to solve, or the pods arrive very slowly so it is not possible to wait to have a large set to make the allocation.

A. BASELINE METHODS

To verify the performance and the proper operation of the proposed methods, we have implemented a set of algorithms that will be used as a baseline for comparison, providing a lower bound of the worst possible performance of the methods. These methods are not feasible to use in a real environment as their behaviour is not suitable at all (e.g., assign nodes randomly), but they serve as a benchmark for the comparison with the rest of the methods as the performance is considerably limited. In the following subsections we will present methods that are applicable in real environments and will serve to compare performance in the experiments.

The first algorithm is the simplest of all, when it receives a set of pods to assign directly it chooses the node randomly, therefore its performance will be the worst of all as most of the time the final assignment will violate a large number of constraints. The result of this algorithm defines the upper bound of worst performance and is therefore useless.

The second algorithm is an improvement of the previous one, it is a random method but it tries to respect as much as possible the constraints of the problem when choosing a node. In this way, the result is quite acceptable as it limits the number of restrictions that are violated but it is not the best as it does not optimise the problem, it only randomly selects a node from the set of possible nodes as a solution. The algorithm is for comparison because it provides a way to see if the proposed methods are able to offer superior performance in choosing better nodes within the set of nodes validated as a solution, if there is no improvement it would be concluded that the methods are not worthwhile as they do not improve on the simpler case of randomly assigning valid nodes. Furthermore, it is important to mention that this method is not feasible as its computational cost is very high and it is only used for comparison purposes in the results analysis section.

On the other hand, another algorithm that is implemented is a basic Round-Robin algorithm that essentially assigns the pods in order to each node. This method does not consider any of the constraints of the problem and serves to simulate the behaviour that the original K8s scheduler would have, since if it does not include the Continuum metadata, it would just assign the pods to the nodes following a round-robin scheme.

With these three algorithms we therefore have a suite of methods to determine respectively the worst possible result, an acceptable result and a result similar to native K8s.

B. POD-POD ALLOCATION

The first type of algorithms are those that perform the allocation following the pod-to-pod style of Kubernetes, therefore they individually assign the pods in the queue one by one without attending to the global result of the mappings.

As already mentioned, the first two methods are exclusively for defining the worst possible performance as their assignment is purely random. However, the constraint-aware Random algorithm gives an acceptable result by ensuring that the solution satisfies the constraints of the problem regardless of the end node. The third method that will be used in the tests is Round-Robin, a very common method in networks because in systems where the elements are similar and there is no clear preference between each of them, it achieves a good result by distributing the work equally, so each pod in the queue of pods to be assigned will be allocated to each node in order in a cyclic pattern. The results of this method will be used to simulate the default behaviour of K8s, because if no specific metric is defined, the Kubernetes scheduler will distribute the pods between the nodes in an orderly fashion.

Finally, we have implemented a greedy method for the assignment of the pods to the best nodes, the methodology consists of assigning a score to each node (using the cost function of the eq. 4 and penalising the nodes that do not meet the restrictions) and then ordering the nodes and choosing the one with the lowest cost. In this way a fast pod-pod assignment is achieved respecting the constraints of the problem and with an acceptable sub-optimal result, since in many types of problems greedy algorithms are not able to give the optimal solution and a more complex or full exploration method is required [25], [26].

Although the real objective is to achieve allocations following a Batch scheme, pod-pod allocations are useful to obtain fast solutions or to implement more complex mechanisms such as multi-objective optimisations, for example, selecting the best node according to its latency and consumption using a Pareto front.

C. BATCH ALLOCATION

To obtain a result closer to the optimal one, it is mandatory to process the allocations in batches in order to take into account the joint allocation decisions, so that the best possible combination can be found. Hence, the optimisation problem defined above must be solved.

As an approach to solving the optimisation problem previously defined (eq 2), given that we want to find the best possible solution, we have opted for a strategy of full exploration of the state space. For this, we have implemented an algorithm based on backtracking, with some optimisations to make the exploration faster, in such a way that the algorithm will test all possible assignments of nodes to the list

of services to be deployed. The method essentially tests each set of pod-node assignment combinations (the array defined in eq. 1) in an orderly fashion and checks whether they meet the constraints (restrictions of the problem in eq. 2) and what fitness (eq. 4) it gives, writing down the best combination it finds. However, it is important to mention that it is a method that usually has a high and exponentially growing computational cost. Therefore, if the problem is of small size (5-10 pods and 10-20 nodes) its use is feasible, but if a large number of pods have to be assigned, its use is prohibitive because its execution time is very large.

For large scenarios, the allocation can be split into smaller batches, slightly reducing the performance of the final allocation but making it manageable, for example, by separating one allocation of 20 pods into two batch of 10 pods. On the other hand, another alternative solution in this case is to use methods based on metaheuristic techniques such as Genetic Algorithms [27] or Particle Swarm Optimisation [28]. Thus, even if the problem is very large, using population-based techniques it is possible in a short time to find solutions with a performance relatively close to the optimal one. To demonstrate this alternative, we have implemented a genetic algorithm (similar to that proposed in [29]) that solves the node assignment problem through a population of individuals that are iteratively improved until reaching a solution very close to the optimal one.

V. SIMULATIONS RESULTS AND DISCUSSION

In this section the performance of the proposed solution is analyzed using the results obtained in a real test environment. We will also evaluate the results against other methods described above to compare and prove the necessity of replacing the native Kubernetes scheduler to satisfy the requirements of the Computing Continuum. Furthermore, the results analysis section will focus on the demonstration and justification of the need to adapt the pod-pod K8s scheduler for the Continuum, so it will not be intended as a benchmark between different allocation algorithms.

A. ARCHITECTURE

The architecture of the test scenario is the already mentioned Computing Continuum, consisting of different layers of devices further and further away from the end user but with higher computational capacity. Figure 3 summarises the one we have used for testing, in which we have divided the devices into three levels.

The first level (Edge Layer) is the closest to the users and consists of a set of low-power, low-computing-capacity devices that are close enough to the users to provide low-latency, high-bandwidth access to services.

The second level (Fog Layer) is a mid-point of the architecture where more powerful servers are deployed to handle all the heavy tasks that the edge devices are unable to tackle. These devices are usually relatively close (same region or state) as the edge devices so their latency is not very high and have an acceptable bandwidth.

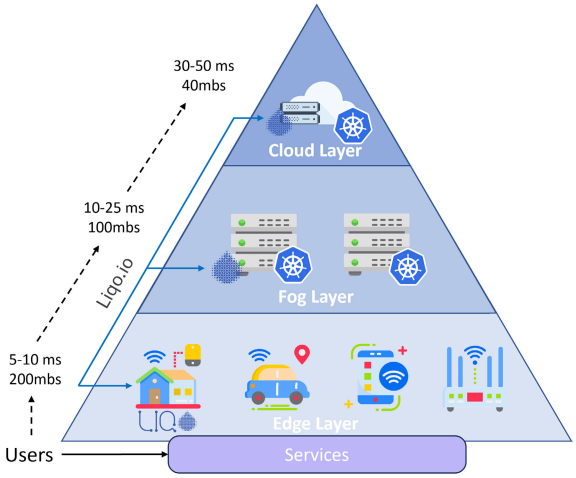

FIGURE 3. Computing Continuum Architecture (Edge-Fog-Cloud Schema).

Finally, the third level represents the Cloud which consists of large data centres with high computing capacity. The disadvantage of the Cloud is that due to the small number of nodes and the great distance from the end users, its latency is usually very high and its bandwidth very low.

In our tests the Edge layer is designed to have a latency between 5 and 10 milliseconds while maintaining a bandwidth of 200mbps. The Fog layer provides a latency between 10 and 25ms and a bandwidth of 100mbps and lastly the Cloud layer has a latency of 30 to 50ms and a bandwidth of 40mbps.

To connect the nodes, deployed as independent Kubernetes clusters, to each other we have used a very promising new technology to interconnect clusters simply and seamlessly known as Liqo [30]. The main idea of the toolkit provided by the open-source Liqo.io project is to be able to integrate several Kubernetes-based clusters in a completely transparent and user-friendly way for the administrator, simulating the remote clusters as virtual nodes. Also, in order for the orchestrator to be aware of the status of each node, it is obligatory to register the node (and properly authenticate itself using secure techniques) in the infrastructure first, so that the orchestrator is properly informed of the hardware characteristics, capabilities and metadata of the nodes to be considered in the resource management process. An example of a mechanism for offering and registering resources from nodes to users who want to use them is the one proposed by Galantino et al. in [31], where they present a protocol (REAR) for the management of request management and resource reservation in a Continuum Computing network.

Regarding the operation of the application and service deployment process, the first step is to receive the deployment request (set of services) in our orchestrator which will collect the infrastructure information and will solve the allocation problem considering the constraints and optimising the result. After that, it will send the request to Kubernetes to deploy the pods and send them to the pod-to-node queue, being processed by our custom scheduler which will assign the pods

TABLE 1. List of cloud, fog and edge nodes.

	Hannover	Turin	Madrid	Valencia	Murcia	Edge1	Edge2	Edge3	Edge4	Edge5
milliCPU	8000	10000	6000	6000	6000	4000	4000	4000	4000	4000
RAM	8 GB	10 GB	4 GB	4 GB	4 GB	2 GB	2 GB	2 GB	2 GB	2 GB
Available CPU	2768	4250	2457	3900	3120	2500	650	2500	1200	3120
Available RAM	6656	7680	3584	2304	3072	1792	1664	1920	1280	1408
CPU Speed	31500	43500	17500	19000	21500	7000	10200	8500	9500	11000
Location	Germany	Italy	Spain	Spain	Spain	Spain	Spain	Spain	Spain	Spain
Latency	40ms	55ms	15ms	20ms	25ms	5ms	7ms	8ms	9ms	10ms
Avg Network Delay	4552ms	3538ms	2125ms	1800ms	1535ms	500ms	650ms	800ms	900ms	1000ms
Bandwidth	50 mbs	40 mbs	80 mbs	90 mbs	100 mbs	210 mbs	180 mbs	190 mbs	160 mbs	190 mbs
Capabilities	RAM-ECC GPU AES-NI	RAM-ECC GPU AES-NI	RAM-ECC GPU AES-NI	RAM-ECC GPU AES-NI	RAM-ECC GPU AES-NI	AES-NI	AES-NI	AES-NI	AES-NI	AES-NI

to the nodes decided in the first step. Finally, the orchestrator is notified that the process has been completed so that the user or administrator can be informed.

Moreover, the orchestrator will periodically check the status of the infrastructure and collect all the statistical information required, for example, CPU and RAM usage of the nodes, latency between layers, among others.

B. SIMULATION SETUP

To perform the tests in a realistic Kubernetes environment, 10 virtual machines were deployed in different locations and each one was configured to host a Kubernetes cluster. Each virtual machine has been modified to have the defined CPU performance, latency and bandwidth using the cpulimit [32] and Traffic Control (tc) [33] tools. The implementation of our custom scheduler is based on the example proposed by sanposhiho [34], and for the pod assignment to be decided with our scheduler we have used the ''schedulerName'' parameter in each Pod. Table 1 fully summarises the technical information of the testing environment. As can be seen, the set of k8s nodes is heterogeneous as they have different computational power, capabilities, latencies and locations. To synchronize the information related to the state of each node with the orchestrator, we have used the metrics server provided by Kubernetes together with a distributed Prometheus server to register all the metrics not supported by K8s.

Table 2 summarises the information related to the test, such as alpha parameters of the cost function and the JMeter stress test specifications.

C. DEPLOYED APPLICATIONS

In order to verify the implemented methods, a group of applications (services) have been designed to be deployed on the nodes. Each application has specific CPU and RAM requirements (in the same way as a standard Kubernetes deploy) but additionally we can add conditions such as affinity, anti-affinity, minimum latency, desired latency, location restrictions, specific hardware capabilities requirements (GPU, RAM-ECC, NVMe disks, AES-NI instructions, ...) and many others. In practice, each application is based on

TABLE 2. Parameters of the simulation.

Parameter	Value
Alpha parameters	{0.8, -1.2, -15}
Genetic Population	100 Individuals
Genetic Generations	200 Iterations
Request per user	60
Request delay	4 seconds
Service Compressor	2 Users
Service Api	8 Users
Service Database	6 Users
Service ClassificR1	4 Users
Service ClassificR2	4 Users
Service WebR1	4 Users
Service WebR2	4 Users
Service WebR3	4 Users

the same docker container which via machine environment variables and Pod parameters is adjusted to procedurally simulate realistic CPU and thread usage similar to those expected. Table 3 details the services we have designed to perform the stress tests of each method on the infrastructure.

TABLE 3. Services to be deployed.

	Web	Database	Api	classificator	compressor
Nº Replicas	3	1	1	2	1
Location	Spain	Italy	Europe	Europe	Europe
Max Latency	15ms	30ms	50ms	-	-
Desired Latency	5ms	-	-	-	-
Anti-Affinity	Web	-	Web Database	-	-
Software Caps	TLS 1.3 GZIP	SQL	TLS 1.3 PartialContent	IMAGENET	-
Hardware Caps	-	RAM-ECC	TEE	RAM-ECC GPU	-

The Figure 4 shows the dependency scheme of the example application that we have designed, which emulates a kind of simplified image classification app. The part closest to the users is the front end of the application (a PWA web page) which must be deployed in Spain (target users) and must have three replicas for scalability reasons. The middle part of the application is the image classification process

(relatively high computational), which must be deployed within Europe to comply with GDPR. In addition, we have three other complementary services; the database which has to be deployed in Germany as it is the headquarters of the company, a public API that requires little computation and finally a service to obtain the complete model of our classifier which will first have to compress it (high computational cost).

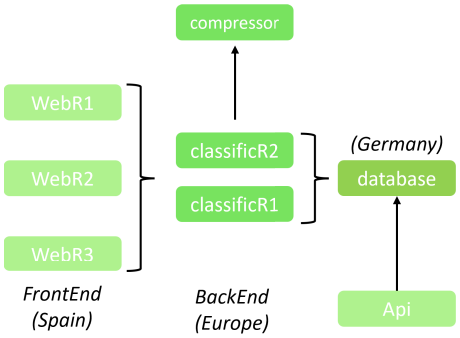

FIGURE 4. Services of the sample application.

Since the application follows the PWA model [35], the services do not interact with each other, it is the end user who will call each service when needed. The image 5 depicts the interaction flow of the application, when the user launches the app it downloads the front end of the web service that serves as an interface to use the App, once the user uploads an image to classify, the user device will connect to the classifier service. Within the application interface there is an option to download the trained model, which will communicate with the "compressor" service to compress the model and download it.

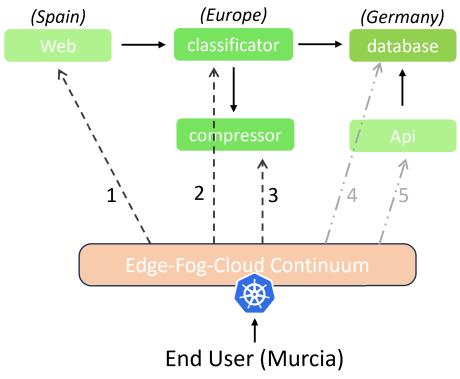

FIGURE 5. Interactions between the user and each service of the application.

D. RESULTS

In this section we will present the results of the tests conducted in a real environment of virtualised nodes, according to the characteristics and requirements specified above. The analysis of these results will focus on justifying the need to replace the K8s scheduler in the context of the Computing Continuum, so different types of algorithms to solve the optimisation problem will not be analysed, as each of them will be suitable in different contexts according to the needs of the administrators.

In order to visually illustrate the operation of the optimal algorithm for allocating batches of services to the infrastructure, the orchestrator is requested to deploy three times the use case presented above, to force the nodes to become saturated and force the algorithm to decide new nodes for the next deploy. Figure 6 shows the process, starting initially from the empty infrastructure (top left), and at each step showing the allocation of the use case services up to three times (bottom right).

The first stage shows the starting state of the nodes that we have defined, showing to the right of each one of them a bar with the percentage of the amount of CPU required by the node's services and below a number with the amount of available milliCPU. In the next step we can see how the set of services of the use case is allocated (using the optimal algorithm) in the infrastructure, highlighting how the database is placed in the node in Germany (location requirement), the heaviest tasks (classify and compress) end up in the fastest nodes (Turin and Madrid) and the rest of the web services are distributed among the nodes with lower latency. It is therefore confirmed that the Continuum requirements are being respected as services are distributed across a multi-level infrastructure in the most optimal way according to latency, computational speed, etc, and also respecting constraints such as location and anti-affinity.

However, to ensure that the proposal works properly and dynamically, it is necessary to force the nodes to reach their maximum capacity so that the method has to search for alternative nodes that respect the constraints. For this purpose, the allocation of the use case services to the system is repeated twice more (without overwriting them), so that some of the nodes are saturated with the extra work. This can be seen in the third stage (bottom left) where the compression service is assigned again to the most powerful node (Turin) taking it to the limit of CPU capacity so that the classification service is distributed between the next two most powerful nodes (Madrid and Valencia).

Similarly, as we continue to the last step (bottom right) it becomes more complicated to allocate services as many of the best candidate nodes are already at their maximum capacity, so compute-intensive services start to be distributed closer and closer to the edge, for example the compression service is sent to the Murcia node as it has the least work and has an acceptable compute power.

If a similar approach to K8s had been followed, it would not have been possible to optimise the allocation of services to nodes in this way, so there is a clear need to enhance the default scheduler, especially for the batch allocation process. Furthermore, the number of services varies based on the algorithm employed and in our situation, the optimal algorithm's high cost results in fewer services, a number that can be readily increased by utilizing evolutionary algorithms. However, to show its potential we have performed a test where we fully optimise the allocation of the use case services

FIGURE 6. Evolution of allocating service batches to the infrastructure (from top left to bottom right).

three times, so the result will be the best possible allocation of the last step in Figure 6. The result is shown in the left part of Figure 7, although it is computationally unfeasible within the scheduler, it is useful to clearly see the good performance of the proposal as it shows how the most expensive services (compressor and classifier) are kept within the most powerful nodes while the web services are perfectly distributed in the edge nodes with lower latency. Likewise, the right-hand side of the figure depicts an example of a pod-pod allocation using the proposed greedy algorithm. As mentioned above, this approach would be a direct improvement on the Kubernetes scheduler as it would consider the continuum requirements and search for the best node to meet the constraints, while maintaining a fast pod-pod allocation approach. But, the solution obtained has a much lower performance than the optimal batch solution because each allocation is done independently and without considering the others, so it tends immediately to saturate the nodes with the best cost function and many nodes are left with a reduced utilization.

After graphically analyzing an example of how the pods would be distributed across the nodes using the proposed methods, we will proceed to describe the second set of tests in this section. We will deploy the services of the use case three times (d1, d2, d3) on the initial infrastructure in the same manner as in the previous test, utilizing each of the described methods: the complete deployment of all the services (Figure 7 left), the deployment of each batch of services individually (Figure 6), the deployment of each batch of services individually using the Genetic algorithm instead of the previous one based on Backtracking, the pod-pod deployment with the greedy algorithm (Figure 7 right), the algorithm that allocates pods randomly while meeting the constraints, a round-robin-based method, and finally, a completely random algorithm. Then, we will measure the result of the pod-node allocation on the virtual node infrastructure by conducting a series of stress tests using the Apache JMeter tool. These tests will assess whether the performance of the deployed services, specifically the average response time, is affected when the load on them is significantly increased. The complete results of all the tests are summarized in Figure 8, showing the average response time for each service and deployment according to the allocation method used. To perform the stress tests, JMeter has been configured with the configuration specified in table 2, in such a way that every 4 seconds a request is generated to each service multiplied by the number of users of that service, thus simulating an intensive use of all the applications deployed on the infrastructure. At the beginning of the JMeter test, nodes are able to resolve requests to services within a few milliseconds, but as the number of requests increases, they become saturated, causing some services to take several seconds to process each request. The final result of the test will be the total average response time of each of the services of each deployment taking into account all the requests made during the stress test.

As shown in Figure 8, the worst result (higher bars, hence longer response time) comes from the totally random algorithm (grey bars), which is obvious since the choice of nodes is not at all appropriate. Moreover, it can be seen how it is not even an optimal allocation in the sense of using available resources properly, since some services (for example, d3-web-r1 and d3-web-r2) have a response time below the rest of the method, this is because they are allocated in very powerful nodes so they are not saturated, but they occupy very expensive resources necessary for other services with a higher workload (compressor or clasificator).

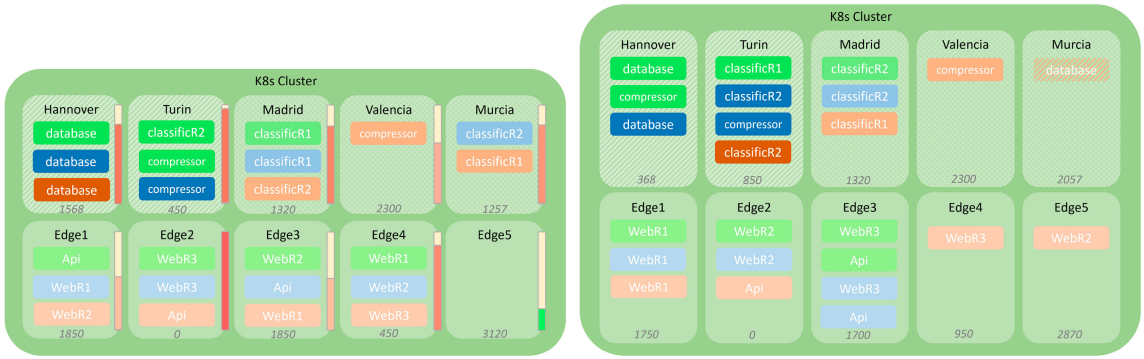

FIGURE 7. Complete Batch deploy vs Pod-Pod Greedy allocation.

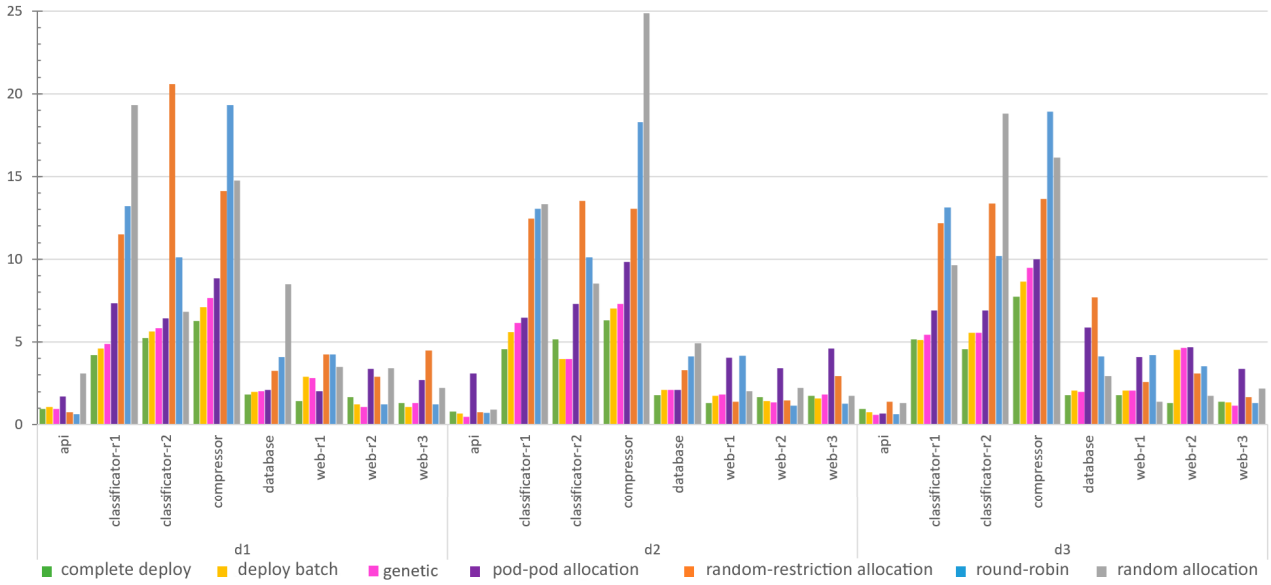

FIGURE 8. Average response time of each service for each method.

Similarly, the round-robin based method (blue bars) has a reduced performance that is slightly higher than the previous method simply because it distributes the services equally among the nodes.

On the other hand, the random method that respects the restrictions (orange bars) improves the response time by considering the problem constraints but in some services (e.g. d1-classif-r2) it gives a bad result as it is still a random method that does not choose the best node. However, as mentioned above, it is a good method to define the baseline performance to be improved, as it takes into account the requirements of the continuum but does not choose the best possible node.

As a first approach to the task of assigning pods to nodes we have the greedy algorithm (purple bars) that follows a pod-pod assignment pattern but takes into account that the best node (lowest value of the cost function) must be chosen from the set of valid nodes (they meet the requirements of the deployment). As can be seen in the graph, in all services the result is acceptable and remains similar between the three deployments, however some services with low workload (e.g. web) tend to have a low performance as many end up in the same nodes as explained previously about figure 7. Since pod-by-pod allocation is done individually it is not possible to optimise the full set of allocations so that some nodes that have a good cost function for a given service end up with several instances of that service as the best candidate appears at each stage, when if all allocations are considered at once it is possible to identify that it is better to deploy them on other nodes even if their cost function is worse but the total cost is optimised.

Adding the possibility of being able to solve the allocation problem in blocks of services (batch) leads to a clear improvement in the performance of applications. The result is seen in the yellow bars representing the method that deploys the pods in three batches to optimise the choice of nodes in each batch using an optimal algorithm. This method gives a much better performance than the other methods, where there is a clear improvement between the pod-pod and

batch schemes, justifying the need to improve the pod-node allocation process so that it can consider several pods at the same time. Moreover, the alternative based on a Genetic Algorithm offers a performance quite similar to the optimal one based on Backtracking, but with a shorter execution time, which allows its application in real-world environments where execution time is critical, even if the solution is not the best of all.

Finally, we perform a test where we run the optimal method (green bars) to perform the allocation of the three deploys as one, thus giving the maximum optimal result of our tests. We can appreciate in the graph how the response time of the services is improved since the allocation is slightly better, but even so the improvement is limited, so it can be inferred that splitting the deploy between three does not have a critical impact on performance and allows it to be computationally possible to solve the optimisation problem by considerably reducing the size of the problem.

In conclusion, while the native k8s scheduler is generally functional, it is not the most suitable scheduler for a Continuum Computing scenario as illustrated by the Round-Robin and Random-Restriction algorithms. Similarly, if it is possible to integrate the continuum requirements into the scheduler, the performance improves directly (random algorithm that respects the constraints and greedy pod-pod algorithm). However, performance will always be limited by the pod-to-pod allocation scheme offered by Kubernetes, so if we want to improve performance we need to be able to allocate several pods at once (batch). This is evidenced by batch-based deployments, where the improvement is clearly seen in situations where the infrastructure is saturated as the choice of nodes has been the most appropriate considering the whole set of services.

VI. CONCLUSION AND FUTURE WORK

One of the most important current tools for container and microservices management is undoubtedly Kubernetes. However, its principal emphasis is on resource management in Cloud Computing architectures, so its applicability is therefore constrained. To increase the dynamism and flexibility of the K8s elements, different tools, frameworks and plugins are provided to increase the features and adapt to new requirements, e.g. the scheduler extension points. Even so, the native scheduler still has limitations that hinder its application in Continuum Computing architectures, such as the lack of considering relevant parameters in heterogeneous architectures (different computing power and capabilities) and the one-to-one approach to the assignment process of pods to nodes.

To solve this problem we propose a framework that considers all the necessary parameters for the deployment of services in a Continuum, as well as providing a set of algorithms for different applications (pod-pod and batches). We then present a series of results to justify the benefits of our proposal and demonstrate the limitations of the basic Kubernetes orchestration, including the reduced performance provided by the pod-pod approach and how a batch-based approach greatly improves efficiency.

Nevertheless, our objective is to justify the necessity of adapting Kubernetes for Continuum, so the main focus of this work is limited to providing the basic principles and tools that can be used as a basis for implementing more complex solutions. Therefore, future work should focus on implementing more complex algorithms to solve the allocation problem, including Genetic Algorithms, Particle Swarm Optimization (PSO), Ant Colony Optimization, and Reinforcement Learning. Additionally, exploring new approaches for Pod-Pod allocation, such as applying neural networks or multi-objective algorithms like the Pareto Front, would be valuable.

REFERENCES

[1] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, fog, or edge: Where to compute?" *IEEE Internet Comput.*, vol. 25, no. 4, pp. 30–36, Jul. 2021.

[2] D. Rosendo, P. Silva, M. Simonin, A. Costan, and G. Antoniu, "E2Clab: Exploring the computing continuum through repeatable, replicable and reproducible edge-to-cloud experiments," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2020, pp. 176–186.

[3] A. Marchese and O. Tomarchio, "Orchestrating serverless applications in the cloud-to-edge continuum," in *Proc. 1st Int. Workshop Middleware Comput. Continuum*. New York, NY, USA: Association for Computing Machinery, Dec. 2023, pp. 12–17, doi: 10.1145/3631309.3632834.

[4] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo, "Adaptive resource efficient microservice deployment in cloud-edge continuum," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1825–1840, Aug. 2022.

[5] J. H. Joloudari, S. Mojrian, H. Saadatfar, I. Nodehi, F. Fazl, S. K. Shirkharkolaie, R. Alizadehsani, H. M. D. Kabir, R. S. Tan, and U. Acharya, "Resource allocation problem and artificial intelligence: The state-of-the-art review (2009–2023) and open research challenges," *Multimedia Tools Appl.*, vol. 83, pp. 67953–67996, Jan. 2024.

[6] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm Evol. Comput.*, vol. 62, Apr. 2021, Art. no. 100841. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221065022100002X

[7] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2131–2165, 4th Quart., 2021.

[8] S. K. Mishra, S. Mishra, A. Alsayat, N. Z. Jhanjhi, M. Humayun, K. S. Sahoo, and A. K. Luhach, "Energy-aware task allocation for multi-cloud networks," *IEEE Access*, vol. 8, pp. 178825–178834, 2020.

[9] F. Faticanti, M. Savi, F. De Pellegrini, and D. Siracusa, "Locality-aware deployment of application microservices for multi-domain fog computing," *Comput. Commun.*, vol. 203, pp. 180–191, Apr. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366423000506

[10] C. Chakraborty, K. Mishra, S. K. Majhi, and H. K. Bhuyan, "Intelligent latency-aware tasks prioritization and offloading strategy in distributed fog-cloud of things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 2099–2106, Feb. 2023.

[11] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, "Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 360–374, Jan. 2021.

[12] J. Tang, G. Liu, and Q. Pan, "A review on representative swarm intelligence algorithms for solving optimization problems: Applications and trends," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 10, pp. 1627–1643, Oct. 2021.

[13] A. Robles-Enciso and A. F. Skarmeta, "A multi-layer guided reinforcement learning-based tasks offloading in edge computing," *Comput. Netw.*, vol. 220, Jan. 2023, Art. no. 109476. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622005102

[14] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–37, Dec. 2022, doi: 10.1145/3539606.
[15] S. Wen, R. Han, K. Qiu, X. Ma, Z. Li, H. Deng, and C. H. Liu, "K8sSim: A simulation tool for kubernetes schedulers and its applications in scheduling algorithm optimization," *Micromachines*, vol. 14, no. 3, p. 651, Mar. 2023. [Online]. Available: https://www.mdpi.com/2072-666X/14/3/651
[16] T. Lebesbye, J. Mauro, G. Turin, and I. Yu, "Boreas—A service scheduler for optimal kubernetes deployment," in *Proc. Int. Conf. Service-Oriented Comput.*, Nov. 1007, pp. 221–237.
[17] A. Marchese and O. Tomarchio, "Network-aware container placement in cloud-edge kubernetes clusters," in *Proc. 22nd IEEE Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2022, pp. 859–865.
[18] A. Marchese and O. Tomarchio, "Extending the kubernetes platform with network-aware scheduling capabilities," in *Service-Oriented Computing* (Lecture Notes in Computer Science), vol. 13740. Cham, Switzerland: Springer, 2022, pp. 465–480, doi: 10.1007/978-3-031-20984-0_33.
[19] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with kubernetes," *Comput. Commun.*, vol. 159, pp. 161–174, Jun. 2020.
[20] S. Böhm and G. Wirtz, "Towards orchestration of cloud-edge architectures with kubernetes," in *Proc. Int. Summit Smart City 360°*, Nov. 2021, pp. 207–230.
[21] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2012. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972238
[22] T. Öncan, "A survey of the generalized assignment problem and its applications," *Inf. Syst. Oper. Res.*, vol. 45, no. 3, pp. 123–141, Aug. 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:1696231
[23] P. Avella, M. Boccia, and I. Vasilyev, "A computational study of exact knapsack separation for the generalized assignment problem," *Comput. Optim. Appl.*, vol. 45, no. 3, pp. 543–555, Apr. 2010.
[24] *Kubernetes Scheduling Framework*. Accessed: Apr. 30, 2024. [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/
[25] J. Mestre, "Greedy in approximation algorithms," in *Algorithms—ESA 2006*, Y. Azar and T. Erlebach, Eds., Berlin, Germany: Springer, 2006, pp. 528–539.
[26] S. Khuller, B. Raghavachari, and N. E. Young, "Greedy methods," in *Handbook of Approximation Algorithms and Metaheuristics*. Boca Raton, FL, USA: CRC Press, 2018, pp. 55–69.
[27] A. R. Tailor and J. M. Dhodiya, "Multi-objective assignment problems and their solutions by genetic algorithm," in *Computational Management*. Cham, Switzerland: Springer, 2021, pp. 409–428, doi: 10.1007/978-3-030-72929-5_19.
[28] C.-M. Lai, W.-C. Yeh, and Y.-C. Huang, "Entropic simplified swarm optimization for the task assignment problem," *Appl. Soft Comput.*, vol. 58, pp. 115–127, Sep. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617302120
[29] M. S. Ajmal, Z. Iqbal, F. Z. Khan, M. Ahmad, I. Ahmad, and B. B. Gupta, "Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers," *Comput. Electr. Eng.*, vol. 95, Oct. 2021, Art. no. 107419. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790621003839
[30] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti, and A. Manzalini, "Computing without borders: The way towards liquid computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 2820–2838, Sep. 2023.
[31] S. Galantino, E. Albanese, N. Asadov, S. Braghin, F. Cappa, A. Colli-Vignarelli, A. Majid, E. M. Fabregas, J. Marino, L. Moro, L. Nedoshivina, F. Risso, D. Siracusa, A. Skarmeta, and L. Zuanazzi, "Building the cloud continuum with REAR," in *Proc. IEEE Int. Conf. Netw. Softwarization*, Jun. 2024, pp. 67–72.
[32] A. Marletta, "CPU usage limiter for Linux," Sourceforge, Tech. Rep., 2012.
[33] M. P. Stanic, "TC—Traffic control," Linux QOS Control Tool, 2001. [Online]. Available: https://arvanta.net/mps/linux-tc.pdf
[34] *Kubernetes Simple Scheduler—GitHub*. Accessed: May 15, 2024. [Online]. Available: https://github.com/sanposhiho/mini-kube-scheduler/tree/initial-random-scheduler?tab=readme-ov-file
[35] D. Fortunato and J. Bernardino, "Progressive web apps: An alternative to the native mobile apps," in *Proc. 13th Iberian Conf. Inf. Syst. Technol. (CISTI)*, Jun. 2018, pp. 1–6.

ALBERTO ROBLES-ENCISO received the B.S. and M.S. degrees in computer science from the University of Murcia, in 2019 and 2020, respectively, where he is currently pursuing the Ph.D. degree. Since 2021, he has been a Seneca Pre-Doctoral Researcher with the Department of Information and Communications Engineering, University of Murcia. His research interests include orchestration, the Internet of Things, edge computing, and energy optimization.

ANTONIO F. SKARMETA (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Murcia, Murcia, Spain. He has been a Full Professor and the Head of the Research Group ANTS, University of Murcia, since its creation in 1995. Since 2014, he has been the Spanish National Representative of the Marie Skłodowska-Curie Actions within H2020. He has worked on and coordinated different European Union research projects in the Internet of Things (IoT) area, such as SMARTIE, SOCIOTAL, IoT6, and IoTCrawler. His research interests include the integration of security services, identity, the IoT, and smart cities.