Computing (2023) 105:2037–2059
https://doi.org/10.1007/s00607-023-01153-1

SPECIAL ISSUE ARTICLE

Continuous QoS-aware adaptation of Cloud-IoT application placements

Juan Luis Herrera[1] · Javier Berrocal[1] · Stefano Forti[2] · Antonio Brogi[2] · Juan M. Murillo[1]

Received: 30 May 2022 / Accepted: 10 January 2023 / Published online: 28 January 2023

Abstract

Cloud-Internet of Things computing paradigms call for novel and efficient methodologies to decide where to place application services in continuity with Continuous Integration/Continuous Deployment pipelines and infrastructure monitoring. In this article, we present Continuous Adaptation (CA), a new DevOps practice for (1) detecting runtime changes in the application requirements or the infrastructure that, due to their change in resource consumption or their effects on the Quality of Service (QoS), can affect the validity and dependability of the current application placement, and for (2) locally intervening on them by suggesting new placements that ensure all (functional and non-functional) application requirements are met. We assess a prototype of CA, ConDADO, and analyze its performance over a motivating use case. ConDADO adapts the application placement to environmental changes through the use of continuous reasoning, reducing the size of the problem to be solved to optimize its performance. The evaluation shows that ConDADO is able to obtain nearly optimal QoS up to 4.2× faster than alternative techniques, also minimizing the cost of service migration.

✉ Juan Luis Herrera
jlherrerag@unex.es

Javier Berrocal
jberolm@unex.es

Stefano Forti
stefano.forti@unipi.it

Antonio Brogi
antonio.brogi@unipi.it

Juan M. Murillo
juanmamu@unex.es

1 Universidad de Extremadura, Badajoz, Spain

2 Department of Computer Science, University of Pisa, Pisa, Italy

Keywords Adaptation · Cloud computing · Fog computing · Edge computing · Microservices architecture · Internet of Things · DevOps

Mathematics Subject Classification 68N01

1 Introduction

The Internet of Things (IoT) can computerize real-world processes into cyber-physical ones, transforming real inputs and outputs into their digital equivalent. The next generation of IoT applications includes critical and dependable applications, such as in healthcare [1], industry [2], or smart cities [3]. In these applications, the criticality and dependability are reflected by strict Quality of Service (QoS) requirements, which are not trivial to meet. Therefore, we define a dependable application as one where the QoS is sufficient for it to work properly under a given scenario.

In this article, concretely, we focus on the minimization of response times, as performance is a key dependability issue. The cloud used to be the most popular paradigm to deploy IoT applications [4]. Nonetheless, the large distance between final IoT devices and cloud data centers complicates the achievement of these QoS requirements [4]. Thus, recent research is focused on the use of Cloud-IoT continuum paradigms, e.g., edge, or mist computing [4]. Furthermore, these paradigms are often combined with each other, as well as with cloud computing, and as such, the Cloud-IoT continuum epitomizes a multi-paradigm infrastructure. In this multi-paradigm environment, computing nodes that are closer to users can be leveraged to perform some computing tasks, thus reducing the response times from and to the data sources and reinforcing the application's dependability.

In this context, the existence of a wider variety of possible application placements, as well as the changing conditions of the continuum, imply that the management of the deployment and application placement in the Cloud-IoT continuum is also more complicated. For instance, a crashed node, an overloaded server, or network congestion can make an initially optimal application placement not as suitable for the current situation. Indeed, the problem of placing a multi-service application onto Cloud-IoT infrastructures is challenging and interesting for the scientific community, as testified by recent surveys [5, 6]. Solutions based on Mixed-Integer Linear Programming (MILP) have been proposed in the research literature to address this problem. To this end, in our previous work [7], we have proposed the DADO framework, supporting multi-service application placement and replica optimization. Nonetheless, deploying an application following one of these placements is not simple [5].

In cloud environments, deployments are often automatically managed using DevOps practices [8]. Nowadays, Continuous Integration (CI) and Continuous Deployment (CD) have become some of the most well-known practices in DevOps [9]. Companies such as Meta use CI/CD pipelines, coupled with continuous reasoning (i.e., incremental static analyses), to shorten the delivery time of new features [10, 11]. Furthermore, companies are hiring highly specialized system administrators for the management of application deployments in cloud environments, due to the required knowledge and skills [12].

In the Cloud-IoT continuum, the complexity of managing application deployments grows even larger, calling for even more specialized personnel. Nonetheless, the automation of application management in the cloud brought by DevOps would mitigate the management complexity of Cloud-IoT continuum environments, and therefore, DevOps proposals for IoT are already focusing on the continuum [13]. However, the QoS achieved by the original application may change during the whole application lifecycle. Hence, changes in the infrastructure (e.g., changes in the available computing resources due to the deployment of new applications), as well as changes in the app usage (e.g., increases of the user base) or requirements (e.g., library updates that change the resources needed by application components), may also affect whether the QoS constraints are met or not, possibly affecting the dependability of the system. This can reduce the usefulness of CI/CD pipelines because they are neither triggered by QoS degradation due to infrastructural changes nor able to automatically determine a new suitable application placement [13].

Therefore, the deployment of applications through the Cloud-IoT continuum may require the application placement to be optimized and adapted over time to meet the required QoS [5]. In this context, *adapting* the application placement refers to changing the application placement for it to better suit a new or modified environment. Although this problem may arise in multiple other environments, it is in this continuum where the impact is higher, due to its characteristics (e.g., high distribution, large traffic volume, a high number of nodes). Thus, QoS-aware DevOps systems able to identify services unable to meet their requirements due to environmental changes, as well as determining a new, proper placement for them, are desirable. Such systems can automate manual tasks, such as the identification of the problems' causes or the adaptation of the application placement, assisting the specialized personnel.

In this context, existing QoS-aware application placement systems need to become reactive towards environmental changes instead of re-optimizing scenarios from scratch whenever changes occur. Moreover, the systems must be aware of the previous status of the application placement and the environment, giving them a sense of continuity and detecting changes in the system. This awareness of the previous state will allow systems to become reactive to changes, instead of periodically optimizing the application placement. Continuous reasoning techniques were recently proposed to perform this adaptation [14, 15]. Continuous reasoning allows application placement systems to identify the impact of these changes, as well as determine a new placement for the affected services, capable of restoring their correct functioning. Proposals such as FogBrainX [14] use continuous reasoning to determine new potential application placements without, however, considering any optimization metrics (e.g., the response time of the application). To the best of our knowledge, no existing proposals apply continuous reasoning to QoS-optimizing application placement, nor is migration cost considered part of the placement problem. Nonetheless, there is a need for new practices in DevOps to support QoS-aware application placement throughout the Cloud-IoT continuum, as well as for automatic tools that support these practices.

In this work, we propose a new DevOps practice to support application management in continuity with CI/CD pipelines: *Continuous Adaptation* (CA). It is important not to confuse CA (which is the proposed DevOps practice) with continuous reasoning, which refers to a technique that can be employed by concrete application placement

systems.. CA features the incremental, continuous, and QoS-aware *optimization* of next-gen IoT applications and the *adaptation* of their application placement w.r.t. their target objectives (e.g., response time). Furthermore, we present an open-source prototype of a framework to support CA, Continuous DADO (ConDADO), implemented on top of the DADO framework [7]. ConDADO differs from DADO in that it enables incremental, continuous optimization of application placement through service migration, as well as considering the cost of such migrations. On the other hand, DADO is only able to perform optimizations from scratch, unable to know if a service is migrated or not, nor the cost of the suggested migrations. Moreover, DADO is unfeasible for dynamic optimization due to its high optimization times, whereas ConDADO is faster.

The main contributions of this work are: (i) the proposal of CA as a new practice in DevOps that can be integrated into CI/CD pipelines to continuously review and adapt an application's placement, (ii) the proposal of ConDADO, an enabler to CA that, through the use of continuous reasoning, can adapt an application's placement in a QoS-aware manner, and (iii) the evaluation of ConDADO over an IoT facial recognition use case, in which we conclude that ConDADO can adapt application placements to environmental changes in a more efficient and responsive manner than the original DADO.

The rest of this paper is organized as follows: Sect. 2 introduces the motivation for the QoS degradation and dependability problem. Next, Sect. 3 details CA, while Sect. 4 introduces ConDADO. Sect. 5 presents the evaluation. Finally, Sect. 6 presents related works, and Sect. 7 concludes the paper.

2 Motivation

This section describes a use case to better illustrate the challenges of next-gen IoT applications and the need for CA to achieve QoS-awareness. It is important to note that the model for CA we propose abstracts from the concrete sensors, application components, computing devices, and requests behind their technical characteristics (e.g., CPU, RAM, size of data flows). Therefore, while the use case is based on an IoT facial recognition application to provide a concrete point of reference, CA is applicable to a wider variety of contexts, which can include heterogeneous sensors and applications. Moreover, the use case is not meant to be one of the main contributions of the present work.

The use case presented in this section is based on the proposal of Wang et al. [3], which presents an IoT-based real-time facial recognition application. This application is deployed to a smart university campus. The objective of such an application is to detect any possible intruders that may try to enter the facility without authorization, as well as to ease and automate the check-in and check-out process of authorized personnel. To leverage this application, multiple IoT devices equipped with cameras have been set up at the entrances and exits of the facility. These IoT devices stream the video footage from their cameras directly to the application, which detects their faces, extracts the features of the images, and match them against a registry of known faces [3]. This application is expected to be dependable: it must run at nearly real-time

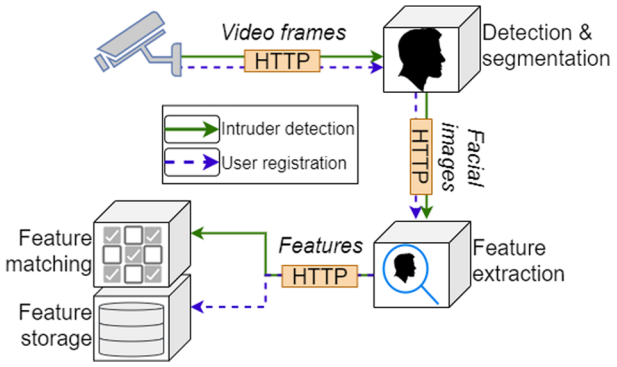


Fig. 1 Architecture of the IoT facial recognition application

performance, i.e. 20 frames per second [3], a strict QoS requirement that must be satisfied at runtime and over time.

Next-generation IoT applications, as well as modern facial recognition applications, tend to follow a microservices architecture (MSA), a design pattern in which the application consists of multiple loosely-coupled modules or *microservices* that collaborate to carry out complex functionalities [3, 5, 16]. Thus, the application is implemented as a set of four microservices, which are depicted in Fig. 1: (i) a face detection and segmentation service that detects if and where faces are in each image frame, (ii) a feature extraction service that extracts the features needed for face recognition, (iii) a feature matching service that compares a set of extracted features with the existing registry, and (iv) a storage service to add new entries to the registry. Each of these microservices can be accessed by IoT devices through the HTTP protocol [16]. Moreover, two main functionalities are supported by the application, the management of authorized users and intruder detection.

In order to perform these functionalities, the IoT devices can request for a *workflow* of these microservices to be executed, in which multiple microservices are pipelined. For instance, to register a new user into the system, an administrator or authorized user can trigger a request from the IoT device for the detection and segmentation microservice to obtain an image that exclusively contains the user's face. Then, the feature extractor is requested and fed the facial image. The resulting features are then added to the registry through the storage microservice. This workflow is depicted by the blue arrows in Fig. 1. On the other hand, to perform intruder detection, the IoT device sends a constant flow of video frames to the application. The video follows a similar pipeline, shown in green in Fig. 1 until the features are obtained. Nonetheless, rather than storing the features, they are fed to the matching microservice instead to recognize the user, as the green line from Fig. 1 represents. The number of requests to these workflows, as well as the computing capabilities and hardware resources of the nodes, affect how many replicas of each of these microservices are deployed, as well as each replica's placement in the Cloud-IoT continuum.

The IoT facial recognition application for this scenario makes use of a DevOps CI/CD pipeline, which follows the schema shown in Fig. 2, which in turn is based on the practices surveyed in [9]. After a planned feature is implemented, the pipeline

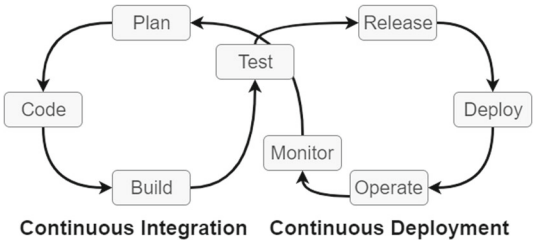


Fig. 2 DevOps CI/CD pipeline

is triggered by a developer pushing one or more *commits* to the *main* branch of the application repository. Whenever the pipeline is triggered, the CI system pulls the changes to the application and builds the new version [9]. If the build is performed successfully, the test suite for the application runs. The built and tested version of the application is then released to CD, which first executes *acceptance tests* over it, whose role is to ensure the application is production-ready. If the acceptance tests pass, the new build of the application is finally deployed to production, and monitored for additional insights (e.g., using multi-paradigm compatible tools such as FogMon [17]).

During the development of the application, in order to ensure the optimal application placement, as well as compliance with the defined QoS requirements, some techniques for design-time application placement optimization, have been proposed, such as [5, 7, 18]. It is important to note, however, that these techniques are meant to be used for the deployment of new releases of the application. Metaphorically, these techniques can only *see a still picture of a live show*: they do not receive information on how the application placement or the environment was prior to that point, and thus, cannot know whether they are suggesting for changes to the application placement or not. Furthermore, the CI/CD pipeline employed by the application is triggered by changes in the code, and not by changes in the infrastructure or the request volume. Hence, the initial deployment, which makes use of this initial application placement, is optimal and dependable, as the QoS requirements of the application are met.

However, most IoT environments, such as this university's infrastructure, are not static environments, and their dynamicity may change the dependability and even the validity of the initial application placement. For instance, in case one of the nodes goes down, or if a commit that updates a library increases the hardware resource requirements of the application, the initial optimization may not hold valid [15]. Furthermore, changes such as a reduction of the available resources due to the unforeseen execution of additional applications may affect the application's QoS, further complicating compliance with the QoS requirements. This motivates the need for a new step within the pipeline, able to be triggered by changes other than those made to the application's codebase and that continuously checks the application placement for the microservices.

This new step of the pipeline requires not only new practices within DevOps but also tools that support them. While one may think that the aforementioned problems can be solved by making use of the same design-time optimization framework, there are two important limitations. On the one hand, these systems often take a long time to yield a solution [7]. While this can be acceptable at design time, during the analyzed events, the

application does not properly work until a new application placement is applied, which is unbearable for most applications. Moreover, design-time optimization frameworks do not receive information about the previous status of the environment, they can only *see the still picture*. Nonetheless, an approach to make existing optimization systems *able to see the live show*, and thus suitable for dynamic environments, is to apply *continuous reasoning* [14, 15]. This approach was originally used in large software repositories such as Facebook: each time there is a change in the repository, an analysis is triggered to statically analyze the parts of the codebase that are affected by the change [10]. Within application placement, continuous reasoning is aimed at finding which microservices of the application placement need to be migrated (i.e., moved between nodes) after a change [15]. Continuous reasoning can be triggered not only by changes to the application's code but also by changes in the available infrastructure resources. Continuous reasoning reduces the size of the application placement problem, as only those elements that were meaningfully affected by environmental changes need to be migrated, which may speed up decision-making [7], and allows for the consideration of migration costs. Thus, through the use of continuous reasoning, existing optimization frameworks can be adapted to CA.

To tame the exponential time complexity of optimization frameworks, it is possible to work on two aspects. On the one hand, we should employ more efficient algorithms to determine the new placement for microservices. On the other hand, we should reduce the size of the problem instance. ConDADO follows both lines, by exploiting state-of-the-art MILP solving techniques in combination with continuous reasoning to promptly respond to environmental changes that affect the dependability and validity of the placement of applications through the Cloud-IoT continuum.

3 Continuous adaptation

To continuously adapt the application placement of next-gen IoT applications to the changes in their dynamic environment, we propose CA, a new practice in DevOps. Unlike other DevOps practices such as CI or CD, CA can be triggered by events that do not directly affect the application's code and check the validity and dependability of the application's placement after the environment changes, appropriately adjusting it if necessary. CA considers the following two kinds of effects that may appear as a result of environmental changes:

> *Broken deployments* are situations in which the existing application placement cannot be used successfully after a change in the environment. For instance, in our running example, the feature extraction deep learning model may be substituted by a more precise one that also consumes more hardware resources, but the application placement may have the microservice running in a device without enough resources to properly run it, or a fog node where some microservices replicas are deployed may fail. In both of these cases, if the application placement is not adapted to this situation, some microservice replicas would be impossible to run, and thus, the deployment is labeled as *broken*. Due to the impact of broken deployments, detecting them is a priority within CA.

QoS violations are subtler than broken deployments and represent dependability issues. In CA, a QoS violation exists whenever, after a change in the environment and if the existing application placement is used, the QoS requirements of the application are not met. While it is possible that the QoS may degrade as a result of these changes in the environment, CA does not deem such situations as QoS violations if the QoS requirements of the application can still be met (i.e., if the application is still dependable enough). Thus, in a QoS violation, the application, or part of it, cannot meet the defined QoS. For instance, the university may need to close one of the entrances of campus to perform maintenance on the area. Therefore, the users that used that entrance to enter or exit the facility will go through other entrances. If that entrance was usually where users were registered, it is likely that no storage microservice exists at other entrances, as replicating the microservice would not improve the response times. However, the existing microservices may not be prepared for the additional burden. Thus, each of the registration requests requires communication with the fog node at the closed entrance. This longer communication increases the workflows' latencies, possibly lowering the performance to less than the required 20 frames per second. The priority of QoS violations is also very high, and thus, they are detected in the same step as broken deployments.

Once either broken deployments or QoS violations are detected, the application placement needs to be adapted to the new environment. For instance, to free some computing resources, some microservice replicas may be migrated from overloaded nodes to others, the microservices from a node that is currently down can be migrated to working nodes, or additional replicas may be deployed to reduce latency. To perform these actions, a new and adapted application placement must be determined, which is the ultimate task of CA. The new application placement must not contain broken deployments or QoS violations, and therefore the effects of QoS generated by the solutions to broken deployments, as well as the validity of the solutions for QoS violations, have to be considered during its determination process. It is also important to consider the migration cost at this point, and thus, when multiple actions can be performed to successfully adapt the application placement, it is preferable to execute those with lower costs.

As a DevOps practice, CA is meant to be integrated with existing CI/CD pipelines. Within this kind of pipeline, CA fits best at the beginning of CD, in a similar manner to continuous reasoning practices such as Facebook Infer fit into CI [10]. In our running example, a new version of the facial recognition application is coded and pushed to the repository. This new version is analyzed for changes using continuous reasoning *à la Facebook Infer*, and the parts affected by the changes in the code are rebuilt and validated using CI. Then, the information about changes to the application and the Cloud-IoT infrastructure is fed to CA, which detects whether a new application placement is required or not using continuous reasoning. Finally, if needed, a new partial application placement for the new release is determined using CA and is executed by CD, as depicted in Fig. 3.

However, for CA to act as intended, environmental changes that do not trigger the standard CI/CD pipeline (e.g., infrastructural changes) need to be able to trigger it as well. Therefore, the infrastructure's monitoring should be able to trigger CA. Further-

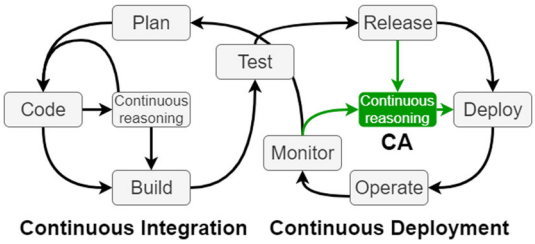


Fig. 3 Integration of CA in the DevOps pipeline

more, the source code may not be directly involved in these changes, and hence, the pipeline can be started directly from CA, as CI would not be necessary. The CA process consists of two main steps. First, the current application placement is scanned to find any broken deployments or QoS violations. Then, both broken deployments and QoS violations are reported and solved by determining a new application placement. The QoS-aware application placement is finally fed to CD and then applied to the infrastructure, closing the CI/CA/CD cycle. Regarding other recently proposed DevOps practices, as well as new practices that may be proposed in the future, we consider two different types of integration with CA. On the one hand, practices that maintain QoS outside of their scope, such as continuous testing [19], do not interfere with CA. As such, these practices can be integrated with CA in a seamless manner. On the other hand, practices that have an overlapping scope with CA may also be integrated with it, although not in such a seamless manner. For example, continuous evaluation [20], which evaluates the performance of the application, can be used to give CA a more precise view of the application's QoS. Furthermore, systems such as FOCloud allow for this performance to also be predicted in an explainable manner [21]. In this regard, the practices must be integrated by either modifying or refining CA's inputs or using CA's output. To better understand the integration of CA in DevOps, as well as its inputs and outputs for its integration with other practices, the integration of CA in a basic DevOps pipeline with CI, CA, and CD as practices is described in Table 1.

Nonetheless, a key to the CI/CD pipeline is the existence of automated tools that support both practices. Furthermore, tools for the automation of the continuous reasoning process, such as Facebook Infer, were also crucial for its integration within CI. Thus, successfully integrating CA into DevOps and enabling CI/CA/CD pipelines requires a framework that supports CA.

4 The ConDADO framework

In this section, we present ConDADO, a CA framework that applies the concepts of continuous reasoning to DADO. DADO [7] is a framework to optimize the application placement and replication of microservices in the Cloud-IoT continuum based on Mixed-Integer Linear Programming (MILP), a technique that provides optimal solutions at the cost of very high optimization times.

Figure 4 illustrates the flow of information and processes used by ConDADO. The execution starts when the monitoring system, which is running in parallel to ConDADO,

Table 1 Summary of the practices of CI/CA/CD pipeline

Practice	Inputs	Description	Outputs
Continuous Integration	Key Performance Indicators (KPIs) of the production environment and requirement backlog	Automated inclusion of the implementation of requirements into the existing codebase, generating binaries for the new version and testing them	New validated application version
Continuous Adaptation	New validated application version or changes in the system's KPIs	Analysis of the validity and QoS of current application placement for the new version or environment and, if required, adaptation thereof	QoS-aware adapted application placement
Continuous Deployment	New validated application version and application placement	Release of the new version installing and maintaining it in the production environment following the application placement, and monitoring its KPIs	Working production environment and associated KPIs

detects a change in the environment, including infrastructure changes or application commits. This monitoring system provides ConDADO with data containing the status of the current scenario, e.g., the QoS and resource requirements of the microservices, the available resources of the nodes, and the workflow requests. The current status data, along with the data of the previous status of the scenario, is fed to the *delta calculator* module. The role of the delta calculator is to quickly assess the differences between both statuses, such as new microservices and workflows, or changes in the existing workflows, microservices, or hardware infrastructure. After such difference is calculated, ConDADO overwrites the previous scenario data with the current one (dashed arrow in Fig. 4) to prepare for the next continuous reasoning optimization step.

Once the difference has been calculated, it is fed, along with the previous application placement, to the *continuous reasoning engine (CRE)*, whose logic is described in Algorithm 1. The CRE performs the first step of CA by detecting broken deployments and QoS violations. As the CRE is called, it looks up the hardware elements whose resources have been negatively changed or that are running microservices that have increased in resource consumption (line 6). This includes the hardware whose resources have become zero, i.e., they are down. Then, the CRE calculates if the application placement is still valid in spite of the identified changes (lines 7-14). If it is not the case, the CRE detects the broken deployment (line 10), and selects and clears the placement of microservices (line 12) until the capacity is high enough (lines 10 and 13). These microservices that have their placement cleared are marked as placement decisions that need to be taken again because they are not valid, a process we call *inval-*

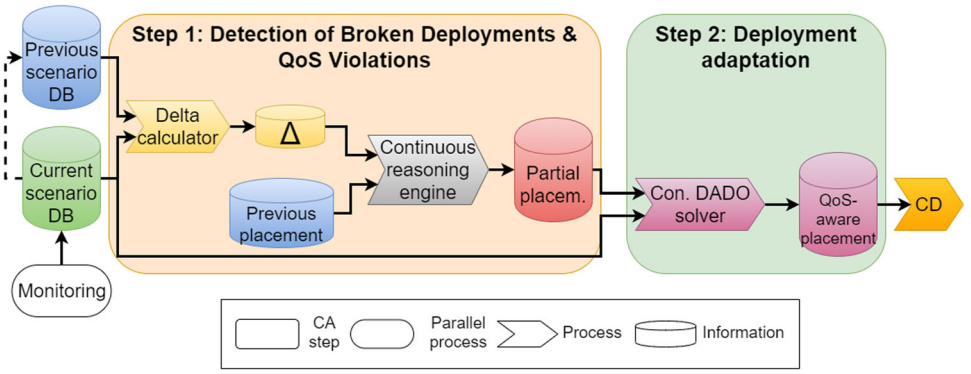


Fig. 4 Bird's eye view of ConDADO

idation. For this process, the CRE invalidates first the placements of the microservices with the lowest migration cost (line 11).

The migration cost of a microservice can be estimated as the size of the package (e.g., Docker image) that needs to be migrated from one machine to another, as heavier microservices require more time to migrate. Nonetheless, the migration cost can be assessed with other criteria, such as the organizational cost of migrating a service (e.g., some services may be costly to migrate because of their criticity for the organization). If multiple microservices have the same migration cost, the CRE invalidates the microservice consuming the most resources first, as it frees more resources to do so, thus minimizing the number of microservices that are invalidated. Finally, the CRE detects QoS violations by listing all the workflows that have not been changed in the delta (line 16), as those that have been changed were already analyzed in the previous loop, and whether any of their placement decisions involves elements whose performance has been deteriorated in the delta (lines 18-21). For these workflows, the CRE calculates whether the performance has worsened enough to be deemed QoS violations (line 19). If so, the CRE invalidates all the application placement decisions of the workflow so they are re-optimized, accounting for the QoS violations that may arise (line 20).

After the CRE runs, the copy of the previous application placement becomes a list of the placement decisions that have not been invalidated (line 23). We label this list as the *partial application placement*. The final step of the CA pipeline is performed by the ConDADO *solver*[1], which takes as input both the current scenario data, as well as the partial application placement. Based on this information, the solver skips over the parts of the model involving decisions that are in the partial application placement. Skipping these operations speeds up the optimization time, as the problem instance is smaller, and thus, it is faster to generate and solve. After the problem is generated, the same MILP optimizer software leveraged by DADO is used to solve it. Once the QoS-aware application placement is obtained, the new migrations, replications, and updated placements can be easily identified by calculating the delta between the original and

[1] The MILP formulation used by DADO is not an original contribution of this paper, and therefore, it is not included. Nonetheless, interested readers can find the detailed formulation in [7].

Algorithm 1 Pseudocode for the CRE

Require: P: *previous deployment*, δ: *differences between the previous and current environment*
1: $invalid := \emptyset$
2: $Nodes := \{n \| n \in \delta.HW \wedge (n.resourceDiff < 0 \vee (m \in \delta.SW \wedge m.resourceDiff > 0 \wedge on(m,n) \in P))\}$ ▷ *Hardware nodes n, where the resources have decreased, or at least a microservice m deployed to it has increased its resource consumption*
3: **for all** $n \in Nodes$ **do**
4: $M := \{m \| on(m,n) \in P\}$ ▷ *All microservices m placed in node n as of P*
5: $required := \sum_{m \in M} m.resources$
6: **while** $required > P(n).resources$ **do** ▷ *While the sum of resource requirements of all microservices m placed in n as of P is greater than the resources of n*
7: $m := extract(M, criteria = (migrationCost, resources))$ ▷ *Extract a microservice m from M following the criteria*
8: $invalid := invalid \cup \{m\}$
9: $required := required - m.resources$ ▷ *Free up the resources taken by m*
10: **end while**
11: **end for**
12: $Workflows := \{wf \| wf \in P.workflows \wedge \notin \delta.workflows \wedge (\exists m \in \delta.SW \wedge m.resourceDiff > 0 \wedge partOf(m,w))\}$ ▷ *Workflows wf that have not been directly modified and request at least a microservice m that has increased its resource consumption*
13: **for all** $wf \in Workflows$ **do**
14: $respTime := getRespTime(wf, P)$
15: **if** $respTime > wf.minQoS$ **then**
16: $invalid := invalid \cup wf.microservices$
17: **end if**
18: **end for**
19: $D := P - invalid$
return D: *valid, partial deployment plan derived from* P, *i.e.*, $D \subseteq P$

the new application placement. Finally, the QoS-aware application placement obtained as a result can be fed to CD, continuing the CI/CA/CD pipeline.

5 Performance evaluation

In this section, the performance of ConDADO is evaluated over multiple conditions of a use case to validate the usefulness of CA in CI/CD pipelines.[2] The evaluation use case, the performed experiments, and the evaluation objectives are detailed in Sect. 5.1. The results obtained by each of the experiments are analyzed in Sects. 5.2, 5.3, and 5.4.

5.1 Evaluation setup

To evaluate ConDADO, an environment based on the facial recognition application from Sect. 2 has been defined. Following this application, IoT devices with cameras can request two kinds of workflows: facial matching and user registration. While registration workflows are triggered by an administrator, application-wise, they are

[2] ConDADO, along with the experiments' code and data is open-source and freely available at https://bitbucket.org/spilab/condado.

requests of microservice execution that come from an IoT camera, and hence, they are modeled in this manner.

The placement of this IoT application is evaluated over two different scenario sizes: small, with 5 IoT cameras and a single fog node, and large, with 15 IoT cameras and 3 fog nodes. The IoT cameras are based on Texas Instruments CC2538 microcontrollers, which connect to the network using 6LoWPAN. The network itself is comprised of switches that connect using Gigabit Ethernet, the same technologies fog nodes use for connection. The fog nodes are based on the instances of the EDGE.NETWORK provider, while the cloud node is based on an AWS m5.xlarge instance. Moreover, the networking details such as latencies and bandwidths are also based on these protocols, as well as real metrics. Regarding the number of microservice replicas, they depend on the exact solution given by each solver. Nonetheless, if we account for the maximum number of replicas, up to 15 microservice replicas can exist in the small scenarios, and up to 45 can exist in the large scenarios. Approximately half of the IoT cameras in each scenario request each type of workflow, and in the cases with odd cameras, there is an additional facial matching workflow. In all cases, the scenarios are optimized with DADO first, and then ConDADO is used to adapt the application placement in a QoS-aware manner. All the experiments were executed with an Intel i7-8565U CPU, with 16 GB of RAM. The MILP solver used is Gurobi[3]. Both DADO and ConDADO are implemented in Python using the *mip* library. The numerical QoS-related results are obtained using DADO's QoS analysis tools [7]. On the other hand, the results related to speed-ups and optimization times are not obtained from these simulations, but from the real timings yielded by both DADO and ConDADO during the evaluation.

Overall, the evaluation has three objectives:

- To validate the usefulness of CA in next-gen IoT applications, consequently validating ConDADO as an enabler for CA (Sect. 5.2)
- To evaluate the differences in the QoS achieved by ConDADO w.r.t. the original DADO (Sect. 5.3), and
- To assess the migration cost and the speed-up of CA solutions compared to the traditional optimization approach (Sect. 5.4).

To achieve these objectives, we have analyzed the results of different scenarios, in order to analyze of the effect of parameters such as the size of the scenario or the impact of the changes. Concretely 20 experiments of two kinds have been tested:

- In the broken deployment experiments, up to 4 microservices and all their replicas experiment an increase of up to 100% in the resources they consume, which may lead to broken deployment situations. These experiments simulate different kinds of changes in the application that could lead to such increases (e.g., the use of more accurate and complex matching techniques, an improvement in video quality, and more detailed video pre-processing).
- In the QoS violation experiments, for cost-effectiveness reasons, we assumed the university increases the resources of their nodes and changes to a less costly cloud provider, which increases the cloud latency by 5 ms.

[3] https://www.gurobi.com/.

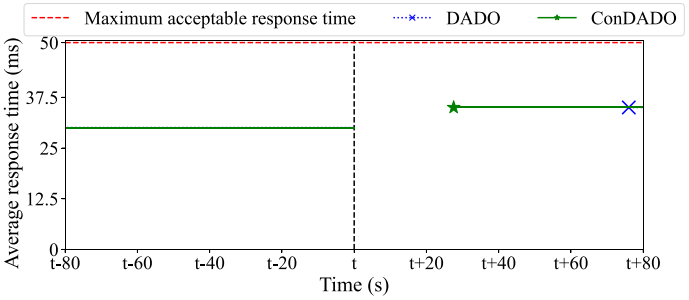


Fig. 5 Continuity in the application's QoS in the large topology

Both experiments analyze the usefulness of CA by comparing the QoS obtained over time, generating a change in the environment at a given point and comparing the gap between the moment a dependable QoS is obtained with and without CA. In order to use a baseline for the usage of non-CA methods, we assume that the original DADO is used to re-optimize the application placement and that it is triggered immediately (i.e., there is no manual interaction). Thus, it is important to note that the gap is incomparable to the gap caused by manual re-optimization and management. Moreover, in order to better show the effects of the use of CA in terms of migrations, all microservices are considered to have a migration cost of 1.

5.2 CA applicability

The first analysis compares the continuity in the application's QoS in the large topology, in a scenario in which the resources consumed by a microservice are increased by 80%. In this analysis, the environmental change occurs at a given instant t. Due to the increase in consumption, the prior deployment becomes broken, and thus, there is a need to determine an adapted application placement. We consider a deployment becomes broken if its response time is greater than 50 ms, as the QoS requirement for the facial recognition application is to have 20 frames per second. This requirement is represented by the dashed red line labeled *Maximum acceptable response time* in the upcoming figures.

The analysis is depicted in Fig. 5, with the time relative to t shown in the X-axis, and the application's QoS can be seen in the Y-axis. The empty gap represents the application's downtime due to the broken deployment, which is of 27.5 s if ConDADO is used to optimize the application placement. On the other hand, the use of DADO requires 76 s to complete, a difference that is visually represented as the gap between ConDADO's marker ($*$) and DADO's ($\times$). Furthermore, both ConDADO and DADO achieve the same QoS with their new application placement. The gap between DADO and ConDADO is of 48.466 s, i.e., ConDADO is almost three times faster than DADO at suggesting new application placements in this scenario (viz., 2.76$\times$). Moreover, the application placement determined by ConDADO also obtains a similar QoS to that obtained with DADO's application placement, further motivating its use for CA, and providing smoother and faster decision-making.

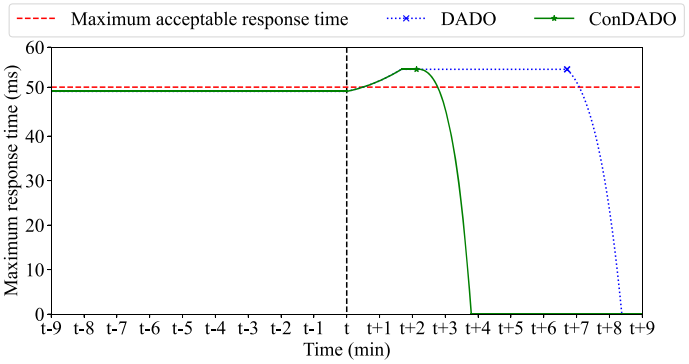


Fig. 6 Evolution of the application's response time in the QoS violation experiment, large topology

The analysis depicted in Fig. 6 is aimed at evaluating ConDADO's ability and efficiency in resolving QoS violations. The first change in the topology is an increase in the resources of the nodes, which did not create any QoS violation or broken deployment, and thus, did not trigger CA. Nonetheless, as the latency in the cloud was increased at moment t, the response time quickly increased, triggering CA. It is important to note that CA was triggered at the peak of the increasing curve, as the shape has been interpolated. In the case of ConDADO, this implies that, at the peak, the delta calculator triggered the rest of the pipeline. Once at this peak, ConDADO determined the placement and migrated the services from the cloud to the fog nodes in 27.5 s, as depicted by the star marker. Conversely, DADO required over 5 minutes (302 s) to perform the same task, as it was not designed to solve QoS violations. Overall, the results show that ConDADO is better suited to adapt to dependability issues such as QoS violations, experiencing a speed-up of approximately 11×.

5.3 QoS with and without CA

The next analysis is aimed at comparing the QoS experienced by the application workflows. The results of this analysis are shown in the form of a boxplot from the large topology, with 4 microservices affected, in Fig. 7. It is important to note that all 18 scenarios involving broken deployments yield very similar results, and thus, only the three highest load scenarios are depicted. Nonetheless, the interested reader can find the data for the response times of workflows in the additional material of the paper. We can clearly split the workflows into two kinds: those that have all of their microservices in nearby continuum devices (i.e., those with nearly 0 response time), and those that have at least one microservice placed on the cloud (i.e., those with 50 ms response time). In the first case, with the 20% hardware increase, we find that DADO is able to bring a few more microservices closer to the IoT devices, as the median is very close to 0. On the other hand, while ConDADO needs to use the cloud more, as shown by the median, the impact on the average response time (depicted as a triangle) is not very large, with under 10 ms of difference. Continuing to the case with an 80% hardware increase, we see that both DADO and ConDADO yield equal

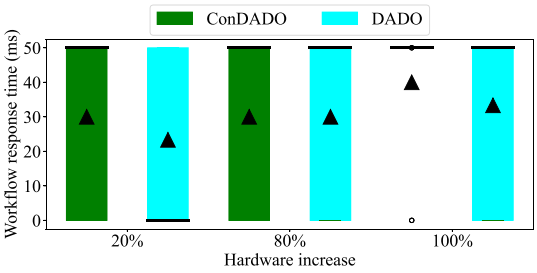


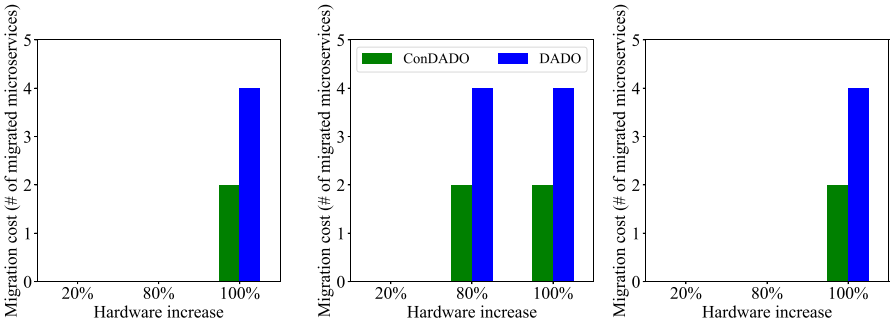
Fig. 7 Boxplot of the workflows' response times, large topology, 4 microservices affected

results, i.e., ConDADO is able to find the same optimal solution as DADO. In the final scenario, ConDADO brings most microservices to the cloud, as the lower point is depicted as an outlier. Nonetheless, we see the difference in average response time w.r.t. DADO is also minimal, similar to the first scenario. Thus, ConDADO is able to provide a similar QoS to DADO for the applications that make use of its suggested placement. Furthermore, the small increase in response time is only experienced by a few users (i.e., workflows), rather than by the application as a whole: the median does not change, those points affected are represented as outliers, and the effects over average response time are minimal. Finally, as every workflow request has a response time of 50 ms or less, the performance requirement of 20 frames per second is fulfilled with the optimized application placements.

5.4 Migration cost and speed-ups with and without CA

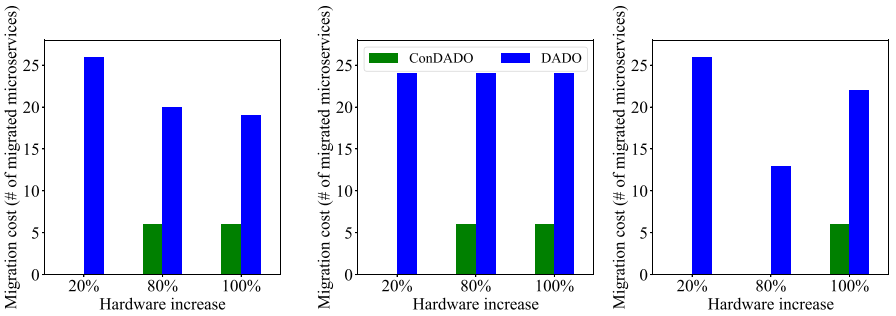
Another key metric of the evaluation is the migration cost achieved by ConDADO's solutions in contrast to those provided by DADO. This metric has been analyzed in both the small and large topology, as depicted by Figs. 8 and 9, respectively. Starting with the smaller topology, in the cases with a single microservice affected (Fig. 8a), we can find that, in some cases (e.g., 20 and 80% hardware requirement increase), the migration cost is 0, i.e., no microservices are migrated. In these cases, the changes due to the hardware requirement increase are mainly a decrease in the number of replicas of certain microservices, and thus, no microservices are strictly *migrated*. Similar results are obtained when 2 (Fig. 8b) or 4 (Fig. 8c) microservices are affected.

On the other hand, in cases where microservices are migrated (e.g., 100% hardware requirement increase in Fig. 8c), ConDADO's solutions have half the migration cost of DADO's. These results are the consequence of ConDADO's CRE, which tries to migrate a small number of microservices, while DADO's lack of CRE allows it to migrate any number of them. Moving to the large topology (Fig. 9), the reduction of migration cost is even higher. As the larger size of the topology implies a higher number of microservices deployed, DADO is more likely to try and migrate them, while ConDADO follows a similar approach to mitigate the migration cost. Focusing on Fig. 9a, in the case with 20% hardware requirement increase, ConDADO simply removes some replicas, while DADO migrates 26 of them, out of the existing 38. Even when ConDADO needs to migrate microservices, such as the one with the 80% increase, DADO needs to migrate up to threefold the number of microservices. If 2



(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

Fig. 8 Migration costs of DADO and ConDADO, small topology

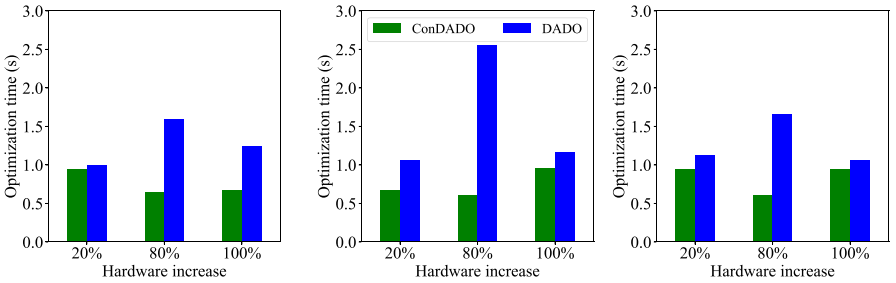


(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

Fig. 9 Migration costs of DADO and ConDADO, large topology

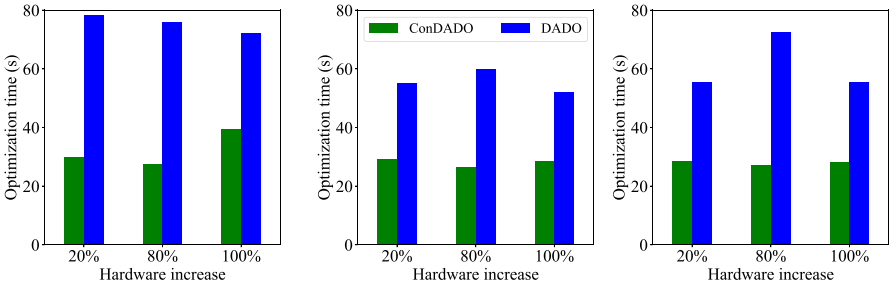
microservices are affected (Fig. 9b), the results are similar, although in general, the difference is even higher (on average, ConDADO migrates 20 microservices less than DADO). Furthermore, the scenarios from Fig. 9c are also the ones depicted in Fig. 7, and thus, this reduction in migration cost has minimal effects on the QoS experienced by the users of the application.

Next, an analysis that compares the times ConDADO and DADO need to optimize each of the scenarios is also performed. The optimization times for the small topology are reported in Fig. 10, while Fig. 11 depicts those of the large topology. Starting with the smaller topology, we see that the times reported by ConDADO are shorter than those from DADO, with ConDADO taking 0.78 s on average, whereas DADO takes 1.38 s. Furthermore, the optimization times from ConDADO are not higher than those from DADO in any of the scenarios. If Fig. 10a–c are compared, it is also possible to see that the number of microservices affected does not have an important impact on the optimization times, and the hardware increase does not have a clear impact either. It is also important to note the case with the 80% hardware increase from Fig. 10b, in which DADO takes 2.55 s to optimize the scenario, while ConDADO only takes 0.60 s.



(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

Fig. 10 Optimization times of DADO and ConDADO, small topology



(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

Fig. 11 Optimization times of DADO and ConDADO, large topology

In the large topology (Fig. 11), the times required by both ConDADO and DADO are higher. Thus, the size of the scenario has an important impact on the optimization time. On average, ConDADO takes 29.41 s to optimize, while DADO requires 64.15 to do so. In this topology, the number of microservices affected does have relevance on DADO's optimization times, as the times from Fig. 11a (on average, 75.52 s) are higher than those from Fig. 11b and c (55.72 and 61.19 s, respectively). While this trend is also followed by ConDADO, its impact is much smaller (32 s on average in Fig. 11a, 28 s on average in Fig. 11b and c. Similarly to the smaller topology, the increase in hardware requirements does not have a clear impact on optimization times. Finally, it is important to note that the results from Fig. 11c correspond with those from Fig. 7, and thus, this reduction in optimization time has very minor effects on the solution's QoS.

Finally, Fig. 12 shows the optimization speed-ups obtained by ConDADO, compared with the original DADO. The reported speed-ups are high, with an average speed-up of 1.93× for the small topology (Fig. 12a) and 2.2× for the large topology (Fig. 12b), and thus, an overall average speed-up of 2.07×. The speed-up tends to grow with infrastructure size due to the fact that ConDADO needs to perform the continuous reasoning step before optimizing, while the original DADO does not. Nevertheless, since the delta calculator and CRE have a smaller complexity than the solver, their burden becomes relatively less significant in larger infrastructures, hence saving more time.

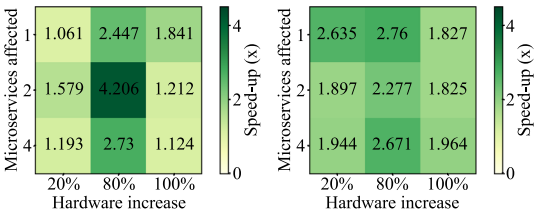


Fig. 12 Optimization speed-ups of continuous DADO w.r.t. original DADO

6 Related work

The classic approach to CI/CD pipelines in DevOps literature is to focus on the delivery of software [9]. Therefore, while CD automatically follows a specified application placement, it does so as a means of delivery to production, and not as a means of meeting the application's QoS requirements [9]. While this approach is adequate for cloud-based deployments, this is not the case in the Cloud-IoT continuum, in which dependability is crucial and each node can provide substantially different QoS. In this section, we briefly summarize the state of the art in the Cloud-IoT continuum and IoT-oriented DevOps and QoS-aware application placement.

DevOps has the objective of shortening the delivery time of new features and allowing for quick reactions to client demands. These features are also relevant in next-gen IoT [13]. DevOps practices need to be adapted to this new paradigm. In [13], Lopez et al. propose an adaptation of the feedback and monitoring from DevOps to the IoT paradigm and multi-paradigm infrastructures. The proposal specifies how to perform fast and continuous monitoring in IoT applications, and can thus be an enabler for CA. Truong and Klein propose the development of DevOps contracts for IoT microservices in [22]. These contracts define the requirements each of the microservices have and are stored in a blockchain ledger. The application developer can write scripts that will be triggered by microservice deployments, requests, or violations of the contract. The main difference between these contracts and CA is their approach to QoS enforcement: DevOps contracts are imperative, as the IoT application developer needs to implement a method to obtain the desired state; while CA is declarative, and the developer only needs to define the desired state. EU projects, such as SODALITE [23], are also working on approaches towards DevOps-integrated systems for the orchestration of application deployment making use of declarative systems, which could be integrated with CA to enable a more fine-grained detail of the deployed microservices and their scaling. Furthermore, SODALITE@RT [24] allows proposals such as SODALITE to be used in the Cloud-IoT continuum.

On the other hand, the obtention of QoS-aware application placements in the Cloud-IoT continuum is currently an open research topic. In [6], Salaht et al. characterize the problem of placing an application's microservices in a dependable way as the Service Placement Problem (SPP). Furthermore, multiple solutions to the SPP are surveyed in this work, including approaches that use different techniques such as integer programming, constrained optimization, or Petri nets. Nonetheless, most of these works are not designed to dynamically adapt the application placement. One of

the works that do consider dynamicity is MigCEP [18]. However, MigCEP's model is oriented to migrating complex event processing microservices to adapt the application placement to user mobility. Changes in the infrastructure or the application are not handled by MigCEP, and therefore, it is not suitable for CA.

Another possibility to consider of dynamicity is proposed by Detour [25]. In the Detour model, each request is analyzed before being sent, and an ad-hoc decision is taken on where the request should be executed. This model requires at least one replica of each microservice to be deployed to every node in order to make this ad-hoc decision. Furthermore, the ad-hoc decision process takes a certain amount of time, which delays every request, and may even become inefficient if the delay introduced by the decision process is higher than the obtained speed-up. Other authors, such as Maamar et al., propose a coordination model for the Cloud-IoT continuum [26]. In this model, the data produced by IoT devices are categorized based on multiple attributes. Depending on these attributes, the model recommends the layer or layers they should be processed at. This approach is fundamentally different from ConDADO, as its model is data-centric rather than service-centric, and thus, it does not specify where to place application services nor how to replicate them.

Another system able to determine QoS-aware application placements is the original DADO framework. DADO was proposed by Herrera et al. in [7] as an enabler for the obtention of QoS-optimal application placements. This framework was meant to be leveraged during the design phase of next-gen IoT applications. DADO supports a wide array of infrastructures and applications, as well as for the consideration of multiple decisions that affect QoS, such as network latency. However, DADO was not designed to adapt its application placements to dynamic environments, and hence, it is unable to know whether a service is migrated or not, or the cost of such migrations. Furthermore, its high execution times are also unsuitable for this task. Thus, ConDADO is an evolution of DADO, adapting its original concepts for this task.

Finally, the use of continuous reasoning to solve the FAPP was initially proposed by FogBrainX [14], a framework oriented to the obtention of QoS-aware application placements, as well as their adaptation to dynamic environments. FogBrainX makes use of continuous reasoning by determining the microservices that are affected by broken deployments or QoS violations and migrating them to suitable servers. While ConDADO takes inspiration from FogBrainX on the implementation of continuous reasoning, there are multiple differences between the frameworks. The main difference is conceptual, as FogBrainX's model is based on the interactions between microservice replicas, while ConDADO's model is based on requests for microservice workflows, allowing ConDADO to replicate and move microservices if required. Furthermore, FogBrainX does not consider optimization objectives, while ConDADO optimize the response time QoS. Thus, while FogBrainX can be suitable for CA, its objectives and model differ from ConDADO's. We believe that, due to these differences, it is very complicated to fairly compare FogBrainX and ConDADO, as the problems they solve, although similar, are fundamentally different.

7 Concluding remarks

The next generation of IoT applications brings computerization to critical real-world processes, and their continuous improvement will be managed using DevOps and CI/CD pipelines. Nonetheless, the criticality of these processes is reflected in IoT applications as strict QoS requirements. In this context, the use of the multi-paradigm Cloud-IoT continuum requires more complex management. Furthermore, the dynamicity of the environment can threaten its dependability. Thus, these continuous changes in QoS motivate the integration of CA as a practice of DevOps, assuring that the QoS of IoT applications is maintained acceptable by adapting its placement.

In this paper, we presented and motivated the concept of CA as a relevant practice for next-gen IoT applications, including its integration within DevOps in CI/CA/CD pipelines. Finally, we presented ConDADO, a framework to perform CA for microservice-based IoT applications in multi-paradigm infrastructures, such as the Cloud-IoT continuum. In the evaluation, ConDADO was shown to be able to provide dependable and valid application placements with a speed-up of up to 4.2× with regard to alternative solutions. CA provides benefits to developers and operators, as it is a framework for automatic optimization of application placement that can be directly integrated into their DevOps workflow. Moreover, ConDADO's continuous reasoning approach allows for the adaptation of the application placement to be highly reactive due to its speed-up w.r.t. the complete optimization approach, as well as to a reduction of the migration costs of each adaptation.

As future work, we expect to extend CA with prototypes able to adapt other aspects of the application's deployment, such as adapting the devices powered on and off to optimize energy efficiency. Moreover, we also intend to create a multi-objective version of ConDADO, able to consider migration cost as an explicit objective, modeling the tradeoff with techniques similar to cost-benefit analysis [27] or cost-benefit at runtime [28]. Furthermore, the cost will be optimized along with additional QoS metrics such as energy consumption. We also expect to integrate ConDADO with existing deployment orchestrators, i.e., Kubernetes. Moreover, we expect to create even faster solutions for CA and *stack* them on ConDADO, allowing CA to react even more quickly to small changes and easy situations, as well as falling back to ConDADO if a suitable solution cannot be found. Last, but not least, we consider evaluating ConDADO over real or emulated network and Cloud-IoT continuum testbeds.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00607-023-01153-1.

Acknowledgements This work was partly funded by Project PID2021-124054OB-C31 (MCI/AEI/FEDER,UE), by the European Regional Development Fund, by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR21133), by the Valhondo Calaff institution, and by project *Energy-aware management of software applications in Cloud-IoT ecosystems* (RIC2021PON_A18), funded with FSE REACT-EU resources by the *Italian Ministry of University and Research* through the *PON Ricerca e Innovazione 2014–20*.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Greco L, Percannella G, Ritrovato P et al (2020) Trends in IoT based solutions for health care: moving AI to the edge. Patt Recogn Lett 135:346–353. https://doi.org/10.1016/j.patrec.2020.05.016
2. Xu H, Yu W, Griffith D et al (2018) A survey on industrial Internet of Things: a cyber-physical systems perspective. IEEE Access 6:78238–78259. https://doi.org/10.1109/ACCESS.2018.2884906
3. Wang S, Zafer M, Leung KK (2017) Online placement of multi-component applications in edge computing environments. IEEE Access 5:2514–2533. https://doi.org/10.1109/ACCESS.2017.2665971
4. Bellavista P, Berrocal J, Corradi A et al (2019) A survey on fog computing for the Internet of Things. Pervas Mobile Comp 52:71–99. https://doi.org/10.1016/j.pmcj.2018.12.007
5. Brogi A, Forti S, Guerrero C et al (2020) How to place your apps in the fog: state of the art and open challenges. Softw: Pract Exper 50(5):719–740. https://doi.org/10.1002/spe.2766
6. Salaht FA, Desprez F, Lebre A (2020) An overview of service placement problem in fog and edge computing. ACM Comp Surv (CSUR) 53(3):1–35. https://doi.org/10.1145/3391196
7. Herrera JL, Galán-Jiménez J, Berrocal J et al (2021) Optimizing the response time in SDN-Fog environments for time-strict IoT applications. IEEE Intern Things J. https://doi.org/10.1109/JIOT.2021.3077992
8. Capizzi A, Distefano S, Mazzara M (2019) From devops to devdataops: data management in devops processes. In: International workshop on software engineering aspects of continuous development and new paradigms of software production and deployment. Springer, pp 52–62. Available from: https://doi.org/10.1007/978-3-030-39306-9_4
9. Shahin M, Babar MA, Zhu L (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access 5:3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629
10. O'Hearn PW (2018) Continuous reasoning: scaling the impact of formal methods. In: Proceedings of ACM/IEEE symposium on logic in computer science, pp 13–25. Available from: https://doi.org/10.1145/3209108
11. Calcagno C, Distefano D, Dubreil J, et al (2015) Moving fast with software verification. In: NASA Formal methods symposium. Springer, pp 3–11. Available from: https://doi.org/10.1007/978-3-319-17524-9_1
12. Hayajneh S, Hamada M, Aljawarneh S (2020) Project management knowledge areas and skills for managing software and cloud projects: overcoming challenges. Recent Adv Comp Sci Communicat 13(3):454–469. https://doi.org/10.2174/2213275912666190429154641
13. López-Peña MA, Díaz J, Pérez JE et al (2020) DevOps for IoT systems: fast and continuous monitoring feedback of system availability. IEEE Intern Thing J 7(10):10695–10707. https://doi.org/10.1109/JIOT.2020.3012763
14. Forti S, Bisicchia G, Brogi A (2022) Declarative continuous reasoning in the cloud-IoT continuum. J Log Comp 32(2):206–232. https://doi.org/10.1093/logcom/exab083

15. Forti S, Brogi A (2020) Continuous reasoning for managing next-gen distributed applications. In: ICLP 2020 Tech. Comm.s. vol. 325 of EPTCS, pp 164–177. Available from: https://doi.org/10.48550/arXiv.2009.10245
16. Rojo J, Herrera JL, Moguel E, et al (2019) A microservice architecture for access control based on long-distance facial recognition. In: International Workshop on Gerontechnology. Springer, pp 219–229. Available from: https://doi.org/10.1007/978-3-030-41494-8_22
17. Forti S, Gaglianese M, Brogi A (2021) Lightweight self-organising distributed monitoring of Fog infrastructures. Fut Generat Comp Syst 114:605–618. https://doi.org/10.1016/j.future.2020.08.011
18. Ottenwälder B, Koldehofe B, Rothermel K, et al (2013) Migcep: operator migration for mobility driven distributed complex event processing. In: Proceedings of the 7th ACM DEBS, pp 183–194. Available from: https://doi.org/10.1145/2488222.2488265
19. Angara J, Gutta S, Prasad S (2018) DevOps with continuous testing architecture and its metrics model. In: Recent findings in intelligent computing techniques. Springer, pp 271–281. Available from: https://doi.org/10.1007/978-981-10-8633-5_28
20. Kao CH (2017) Continuous evaluation for application development on cloud computing environments. In: 2017 International conference on applied system innovation (ICASI); pp 1457–1460. Available from: https://doi.org/10.1109/ICASI.2017.7988191
21. Kumara IP, Ariz M, Chhetri MB et al (2022) FOCloud: feature model guided performance prediction and explanation for deployment configurable cloud applications. IEEE Trans Serv Comp. https://doi.org/10.1109/TSC.2022.3142853
22. Truong HL, Klein P (2020) DevOps contract for assuring execution of IoT microservices in the edge. Intern Things 9:100150. https://doi.org/10.1016/j.iot.2019.100150
23. Di Nitto E, Gorroñogoitia Cruz J, Kumara I, et al (2022) Deployment and operation of complex software in heterogeneous execution environments: the SODALITE approach. Springer. Available from: https://doi.org/10.1007/978-3-031-04961-3
24. Kumara I, Mundt P, Tokmakov K et al (2021) Sodalite@ rt: orchestrating applications on cloud-edge infrastructures. J Grid Comp 19(3):1–23. https://doi.org/10.1007/s10723-021-09572-0
25. Misra S, Saha N (2019) Detour: dynamic task offloading in software-defined fog for IoT applications. IEEE J Select Area Commun 37(5):1159–1166. https://doi.org/10.1109/JSAC.2019.2906793
26. Maamar Z, Baker T, Faci N, et al (2019) Towards a seamless coordination of cloud and fog: illustration through the internet-of-things. In: Proceedings of the 34th ACM/SIGAPP symposium on applied computing; pp 2008–2015. Available from: https://doi.org/10.1145/3297280.3297477
27. Gerostathopoulos I, Raibulet C, Alberts E (2022) Assessing self-adaptation strategies using cost-benefit analysis. In: 2022 IEEE 19th International conference on software architecture companion (ICSA-C). IEEE, pp 92–95. Available from: https://doi.org/10.1109/ICSA-C54293.2022.00023
28. Van Der Donckt MJ, Weyns D, Iftikhar MU, et al (2018) Cost-benefit analysis at runtime for self-adaptive systems applied to an Internet of Things application. In: ENASE; pp 478–490. Available from: https://doi.org/10.5220/0006815404780490