QoS-aware Autonomic Adaptation of Microservices Placement on Edge Devices

Bruno Stévant[1], Jean-Louis Pazat[2] and Alberto Blanc[1]
[1]*IMT Atlantique, IRISA, France*
[2]*Univ. Rennes, INRIA, CNRS, IRISA, France*

Keywords: Edge Computing, Performance Modelization, QoS-aware Placement, Particle Swarm Optimization, Microservice Application Design, Adaptation Strategies.

Abstract: Given the widespread availability of cheap computing and storage devices, as well as the increasing popularity of high speed network connections (e.g., Fiber To The Home (FTTH)), it is feasible for groups of users to share their own resources to build a service hosting platform. In such use-case, the response-time of the service is critical for the quality of experience. We describe a solution to optimize the response-time in the case of an application based on microservices. This solution leverages the flexibility of microservices to dynamically adapt the placement of the application workloads on edge devices. We validate this solution on a production edge infrastructure and discuss possible strategies for the decision rules.

1 INTRODUCTION

Cloud@Home (Cunsolo et al., 2009) or Community Clouds (Khan et al., 2014) are examples of systems of where users contribute with computing and networking resources to a participatory infrastructure capable of processing workloads such as services and Virtual Machine (VM)s. Such systems can provide an alternative to public clouds for users concerned with data privacy and service continuity. Consider, for example, a sport club willing to host community services such as photo-sharing. Using a public cloud provider to host these services is a free and appealing solution, but data is stored by the provider, with potential privacy violations. Furthermore, the provider could unilaterally decide to modify or stop its service.

Participatory cloud solutions propose to use user-owned resources on their premises and federate them in a managed infrastructure comparable to a public cloud. With such solutions, it is possible for a community, such the aforementioned sport club, to create an infrastructure to support its online services with the help of its members. We consider in this work the use-case where some users shares the resources of devices inside their residential network with the community. The services will be executed in the runtime environment created by these devices connected throught the Internet. Data for these services will reside on user devices, thus alleviating privacy concerns. This infrastructure will stay operational as long as community members provide resources.

When using such a solution, the application response-time is a critical parameter for user satisfaction (Egger et al., 2012). Efficient management of such Quality of Service (QoS) parameters is a difficult task in this context because there is no possibility to control the devices and networks. The only way to influence the application response-time is to place the application workloads on devices offering the best performance. Applications based on microservices can help in this endeavor as one can place each microservice independently as opposed to monolithic applications.

In this work, we study a photo-sharing application, decomposed in interdependent microservices as depicted in Figure 1. The *WebUI (UI)* service provides a web photo gallery with thumbnails. The *PhotoHub (PH)* service stores the photos. The *ThumbHub (TH)* service produces on demand thumbnails of photos. The *MetaHub (MH)* service stores photo metadata. We have previously shown (Stevant et al., 2018) how the Particle Swarm Optimization (PSO) heuristic can place each microservice on user-provided devices to minimize the response time (weighted sum of response times corresponding to different user cases, to be precise), in a static case, i.e., where the different parameters (Round Trip Time (RTT), bandwidth, etc.) do not change over time.

Stévant, B., Pazat, J. and Blanc, A.
QoS-aware Autonomic Adaptation of Microservices Placement on Edge Devices.
DOI: 10.5220/0009319902370244
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 237-244
ISBN: 978-989-758-424-4

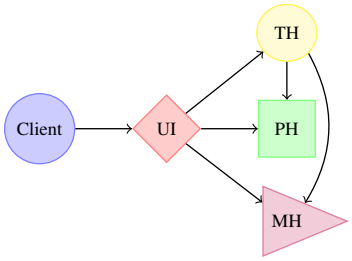


Figure 1: Call graph for the photo-sharing application.

1.1 Problem Statement

In the contribution mentioned above, the PSO heuristic calculates placement solutions from known values of the response-time for each microservice. However these input values may vary according to the availability of computing resources and variations of network QoS parameters (Nishio et al., 2013). These parameters can fluctuate in the context of edge devices connected to residential networks. A solution found by the proposed heuristic will therefore result in an near-optimal response-time but only as long the system's parameters do not change. We started by measuring these parameters (downstream and upstream bandwidth, RTT) for devices connected to various residential networks (Fiber, VDSL) and devices connected to administrated networks (campus lab, Virtual Private Server (VPS) in a datacenter). Figure 2 shows the variations of these parameters over 86 hours. We found that the QoS parameters, especially bandwidths, fluctuate significantly in residential networks because of concurrent usages of these resources from other connected devices.

We characterized the impact of these variations of the network QoS on the response-time of the photo-sharing application by deploying its microservices on the same devices used for the above measurements according to the solution presented in (Stevant et al., 2018). The devices are dedicated to this experiment, therefore computing resources are always available for the application. Figure 2 shows the variations of the response-time of the application and of its microservices measured at the same time as the measurements presented above. We observe that, during a first part of this experiment, while the network QoS parameters are fluctuating, the measured response-time is almost stable. On the last part of the experiment, changes observed in the QoS parameters of a single device (VDSL) after 4000min. resulted in a increase of almost 20% of the application.

We conclude from this experiment that variations of network QoS parameters can have an important impact on the response-time of the application. The deterioration we observed in the previous experiment is sufficient to be perceived by the users. We propose, as a solution to mitigate this, to dynamically adapt the placement according to the variations of the network QoS parameters. A solution for managing such adaptation needs to consider two main issues. The first issue is to detect when an adaptation is needed. The second issue is to specify a suitable alternative placement of the microservices considering the current values of network QoS parameters.

1.2 Contribution

In this article, we describe an autonomic process for adapting the placement of the microservices composing the application according to variations in the parameters influencing its response-time. As a solution to the issues mentioned above, we propose to individually monitor the response time of each deployed microservice. We use these measurements to detect when an adaptation of the placement is needed based on pre-defined decision rules. We use again the PSO heuristic with these measurements to find a new optimized placement of the microservcies.

We validated our approach on an existing participatory infrastructure with devices connected to production networks on which we deployed the photo-sharing application. We integrated probes in the devices to monitor the different response-times and detect any performance change. We tested different decision rules for triggering the adaptation.

2 RELATED WORKS

To identify potential solutions to the mentioned issues, we studied the litterature of three domains closely related to our specific use-case. Different approaches in the field of QoS-aware autonomic resources management in cloud computing are reviewed in (Singh and Chana, 2015). They share with our use case the same objectives of optimizing and adapting the runtime environment of an application according to variations of the underlying infrastructure. Solutions such as (Aryal and Altmann, 2018) and (Khorsand et al., 2018) target different QoS parameters to be optimized, either user-oriented (response-time, availability, throughput) or provider-oriented (consumed energy). We found that these solutions usually consider monolithic or at best 3-tier applications, and static network QoS parameters. These solutions will not fit our use-case, which involves a multi-component application and variations of network QoS parameters.

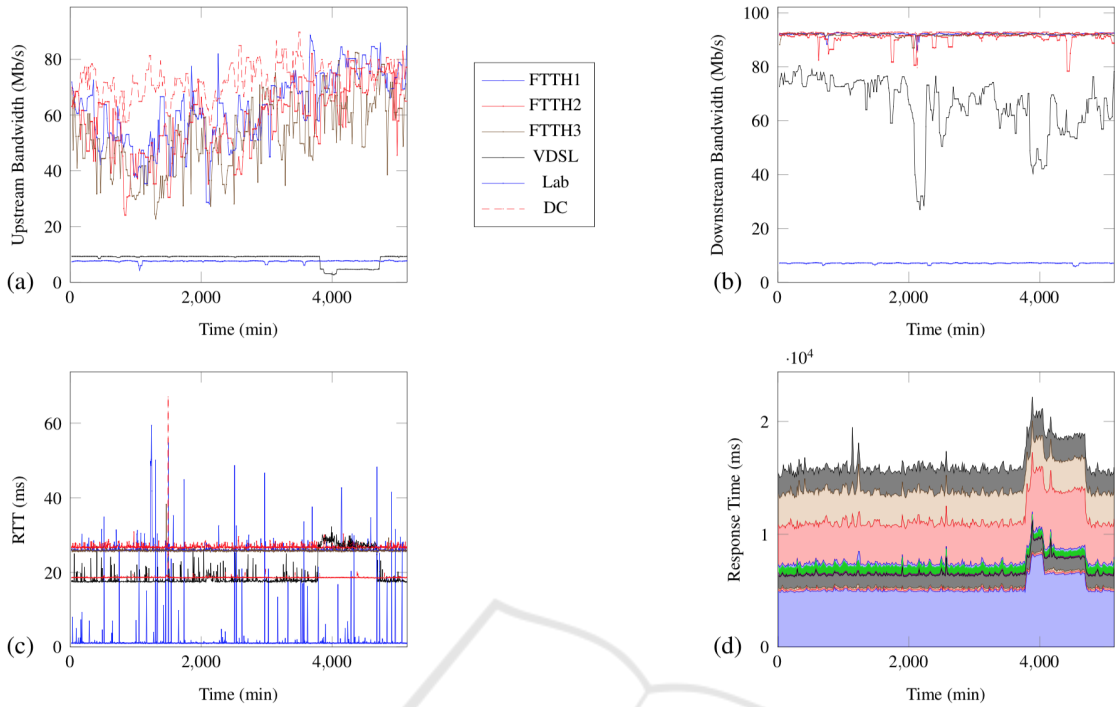


Figure 2: Measured QoS parameters for edge devices and response-time for application using these devices.

A possible option is to consider the architectural model for a microservice based application as a composition of services. We therefore studied Service oriented Architecture (SoA) adaptation solutions reviewed in (Mutanu and Kotonya, 2018). These solutions commonly include network QoS parameters in the criteria for selecting the optimal service location (Cardellini et al., 2017)(Aschoff and Zisman, 2011). These solutions use different strategies for adapting the composition, which try to avoid Service Level Agreement (SLA) violations, either by predicting such problems or reacting to detected violations. To the best of our knowledge, no published work in this area demonstrated how such strategies behave when services are deployed on edge devices.

In (Brogi et al., 2019) the authors explore the problem of placing services in the context of Edge and Fog Computing, and reviewed several solutions capable of dynamically adapting this placement (Skarlat et al., 2018)(Zhang et al., 2016)(Yousefpour et al., 2019). While they consider multi-component applications deployed on edge devices, we found that their adaptation mechanism only react to node arrival, overload or departure, but not to variations of network QoS parameters. Based on these related works, we conclude that our contribution provides an original solution to the problem of QoS-aware placement adaptation of microservices on edge devices.

3 PROPOSED SOLUTION

We describe in this section a solution to optimize the response-time of the application deployed on edge devices, and to adapt the placement of the microservices with the variation of the QoS parameters of the network interconnecting these devices. We propose an approach where the adaptation of the placement is triggered by decisions based on measurements of the response-time of the application. The different functions involved in such decision are structured as represented in Figure 3, and follow the blueprint of an autonomic system defined by IBM in (Kephart and Walsh, 2003). It defines the system as an iterative process with interactions between the infrastructure where the application is deployed, and a decision-making algorithm structured in differents functions.

The *Monitor function* collects response-time measurements from participating nodes. Devices acting as clients of the application measure the global response time of the application. Devices hosting the microservices of the application measure the response-time for function calls between microservices. These different measurements are stored inside *Response-Time (RT) Matrices* for further analysis. The *Analyze function* calculates key indicators from the different measured response-times. The *Decision Rules*, defined by the system administrator, use

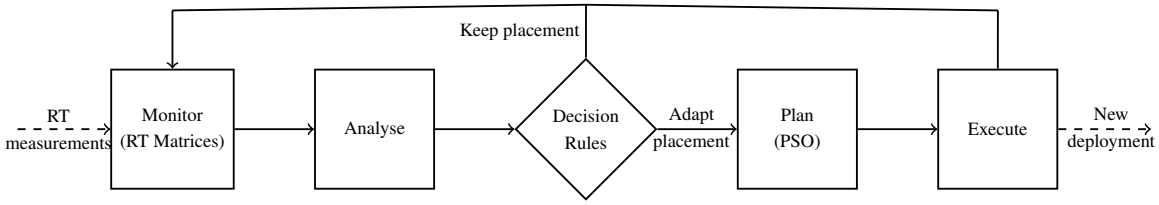


Figure 3: Flowchart for an autonomic adaptation process.

these indicators to decide on the necessity of an adaptation and, in such case, generates an alert. The *Plan function*, triggered by this alert, specifies the adequate actions to be taken to optimize the response-time of the application. This function uses the *PSO heuristic* to calculate a new placement of the microservices from the values stored in the *Response-Time (RT) Matrices*. Finally the *Execute function* deploys the planned placement of the microservices and reconfigure the application.

3.1 Reponse-time Monitoring

The goal of this system is to optimize the response-time of the application from the point of view of the clients. The system monitors the application response time by collecting measurements on the client nodes. These nodes generate calls to the application representing different user actions and measure the time needed by the application to process each of these calls. By collecting and aggregating the response-times of the different user actions measured by all of the clients nodes, the system can compute a (weighted) average value representing the global response-time of the application for the considered clients.

The *Monitor function* collects the response-time of each deployed microservice. Thanks to these measurements it is possible to decompose the application response time as the sum of the response times of each one of the microservices involved in satisfying a user request. The measurements of the response-time for one microservice is stored in the response-time (RT) matrice corresponding to this service. Each response-time measurement is specific to a microservice deployed on a particular node and requested from another node. For the microservice s, $R_s(i, j)$ gives the value for the response-time of service located on i when requested by client j. Once the *Monitor function* has collected sufficient measurements to compute the global response-time of the application, the value is forwarded to the *Analyze function* that will apply rules to decide if an adaptation of the placement is needed.

3.2 Decision Rules

The *Analyze function* decides when adaptation is needed based on the measured response-times and on the rules defined by the system administrator. In this article, we consider two different adaptation strategies.

A first strategy (*reactive*), triggers the adaptation when the monitored values are outside acceptable boundaries . With such a strategy, it is possible for the system to react to a deterioration of the response-time of the application as observed in Figure 2. This reactive strategy should define an appropriate threshold as the least-acceptable response-time for the application. This value can be a fixed number or computed from previous measurements of the response-time.

A second strategy (*proactive*) continuously tries to improve the response-time of the application even if there is no observed deterioration. Indeed, as the network QoS parameters are fluctuating, there is a possibility that an alternative placement of the microservices may result in a lower response-time than the current deployment of the application. This proactive strategy will trigger the evaluation of alternative solutions when a sufficient amount of monitoring information has been collected since the last deployment.

3.3 QoS-aware Placement Algorithm

The algorithm used in the *Plan function* is based on the PSO meta-heuristic presented in (Stevant et al., 2018). A first input parameter for this algorithm is the call-graph of the microservices composing the application, as shown in Figure 1 for the case of the photo-sharing application. This workflow is required to calculate the response-time of the different user requests from the response-time of each microservice. A second parameter of the algorithm is the list of the devices serving as clients of the application. The placement heuristic will find a solution that globally optimize the response-time of the application from the point of view of these clients.

The PSO algorithm searches for a placement among the different possible locations for each microservice of the application. Each placement solu-

tion is evaluated by calculating the expected response-time of the application from values of the RT matrices. The considered placement with the call graph of the application gives the different nodes pairs involved in the network communications implied by the user-actions, hence the values of microservice response-times to be aggregated to estimate the resulting response time. The PSO algorithm is able to find a near-to-optimal solution lowering the response-time in a relatively short time (a couple of seconds).

4 IMPLEMENTATIONS AND RESULTS

To validate our approach in real conditions, we deployed a test infrastructure of nodes connected to residential networks where we can instanciate the microservices of our application. We managed the deployement of the application through the adaptation algorithm described in the previous section and observed during different experiments how the different strategies reacted to the variation of the QoS parameters of these residential networks.

4.1 Testbed

We deployed a testbed using five PC-Engine APU2 small-factor computers and a VPS hosted in a datacenter. Table 1 describes the physical configuration of the devices. A server orchestrates the deployment of the application on designated devices. It remotely manages the activation of the different microservices of the application by instancing containerized images through automation scripts. We used the same scripts in our tests to automate the users requests to the application, from the devices designated as clients.

The physical devices are connected to the Internet through different edge networks. Table 2 describes their nominal values of network QoS parameters. The deployed devices are dedicated to our tests, therefore computing resources are always available for the hoted microservices. However the network between each devices is shared first with the users traffic on their residential network, then globally with other traffic on the Internet. We already have shown the impact of the sharing of this networking resources on QoS parameters in Figure 2.

Table 1: Devices physical configuration.

Device	CPU	Bogomips	RAM
APU2	1x Emb.AMD 1GHz	2000	4 GB
VPS	1x Corei7 3GHz	6400	1 GB

4.2 Monitoring Solution

In addition to the software required for microservices deployment and operation, we integrated in the devices a monitoring tool measuring the different response-times of the services. Each request to a microservice is routed through a transparent proxy that measures the response time for each request and then forwards the result to a monitoring server. This server centralizes all measurements and exposes them to the adaptation algorithm as the Response-Time (RT) matrices described in the previous section.

Before updating the RT matrices, the monitoring server needs to correlate the different measured response-time for each request. This computation is required so that each value of the matrix $R_s(i, j)$ depends only on the QoS parameters between devices i and j. As the proxy is located at the origin of the request, the measured response-time includes the delay for the request to reach the server, for the service to process the request and finally for the response to come back from the server to the client. The transit delay on the network for the request and the reply is dependent of the QoS parameters specific to this device pair. However, in the case of a microservice calling another, the processing time includes the transit delay for a request to a different microservice between another device pair. In such case, the measured response-time should be substracted of the delay implied by this subsequent call before updating the matrix.

To solve this issue, we managed decompose the response-time for a specific request to the application into individual response-times for each involed microservice by first identifying the request with a *corellation ID*. The client of the application injects this identifier as an extra parameter to its request. Each microservice involved in the processing of this request copies the corresponding correlation ID in every following requests. The measurement proxies in the path forward the correlation ID to the monitoring server along with the measured response-times. The monitoring server is then able to correlate the different response-times for each request, using the correlation ID and the call graph of the application.

Table 2: Network QoS parameters of the testbed.

Network	Device	Up/Down BW	RTT
FTTH1,2,3	APU2	100/100 Mbits	30ms
VDSL	APU2	20/80 Mbits	40ms
Lab	APU2	10/10 Mbits	10ms
DC	VPS	100/100 Mbits	20ms

4.3 Methodology for Evaluation

We realized several experiments using the setup described above to evaluate how much different adaptation strategies can perform in optimizing the response-time of the application with actual variations of the QoS parameters in a production network. We have chosen as comparison criterion between strategies, the ratio of adaptations of the placement resulting in an improvement of the response-time of the application.

We implemented the different adaptation strategies inside the *Analyse function* of the architecture. This function is realized inside a single process, configured to apply a single strategy. This process analyzes the monitored response-time and decide if a new placement for the microservices is needed. This process interacts with the monitoring server to analyze the values of the RT matrices and with the server managing the microservices instances to trigger a new deployment of the application. This new process completes the adaptation loop described in section 3.

Each experiment consists first in deploying the photo-sharing application on the test infrastructure. Due to the limited number of available devices, we chose to deploy only a single instance of each of microservices. For this initial deployment, we chose a random location for each instance. We also constrain all deployments to deploy at most one microservice on each device. By forbidding service collocation, we want to maximize the impact of remote communications on the response-time of the application. The RT matrices are initialized before each test with the actual response-times, obtained from a sufficient number of prior tests.

All the devices involved in the test infrastructure serve as clients of the application. We automated the generation of the client requests from the devices in a round-robin scheme: a single device generates sequential requests corresponding to all possible user-actions to the application before another device generates its own requests. The application processes only one client request at a time, without any interference from other concurrent requests. The measured response-time for each user-action of the application is measured by the client and forwarded to the monitoring server. The monitoring server calculates the global response-time of the application once every clients have forwarded their measurements. By continuously generating user requests from the clients, we can observe the variations of the response-time and the impact of changes in network QoS. This value, as well as values in the RT matrices are analyzed by the adaptation strategies.

4.4 Evaluation of a Proactive Adaptation Strategy

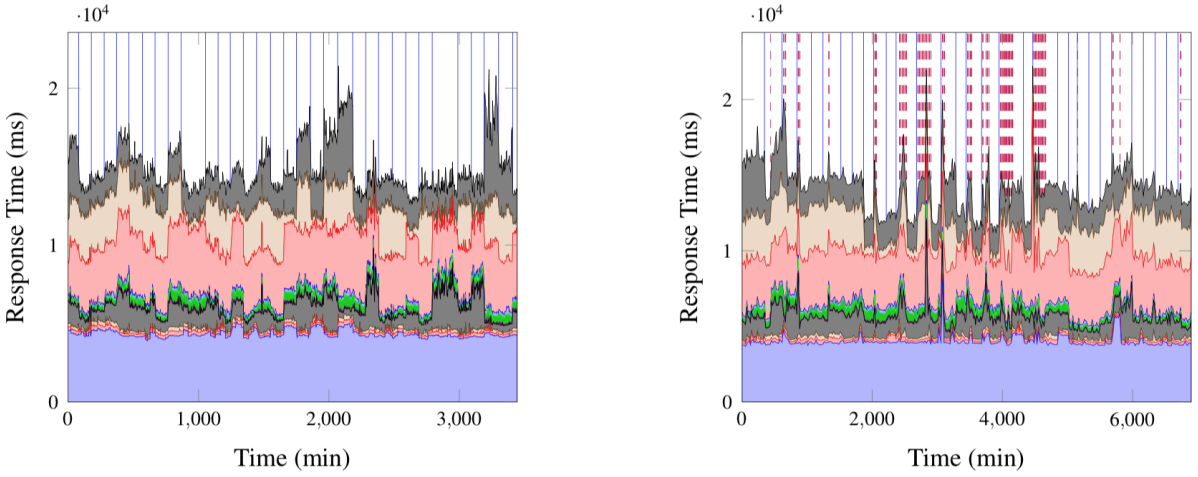
We first implemented a proactive adaptation strategy based on periodic adaptations of the placement of the microservices. This strategy originated from the assumption that variations on the response-time may follow a regular pattern as the volunteers use their networks in a predictable manner. We chose to trigger a new adaptation of the placement of the service every three hours. This time interval roughly corresponds in our tests to the time needed for ten consecutive tests from each client. The requests generated by the clients update the RT matrices with new values for the response-time of the microservices. When triggered, the PSO heuristic will be able to find possible alternative placements based on up-to-date values.

We evaluated this proactive strategy on our test infrastructure through experiments running for a minimum of 50 hours. The observed variations of the response-time of the application of one of the most representative experiments involving the proactive strategy are presented in Figure 4 (a). Adaptations of the placement are represented in this graph using vertical plain lines. We can observe that each adaptation of the placement does not always result in an improved response time of the application. We counted that, over all the adaptations triggered by the proactive strategy in our different experiments, only half of them managed to improve the response-time.

We explain these unsatisfactory results by considering a possible inconsistency between some response-times in the RT matrices and the actual response-times measured with the new placement. Indeed, even if the RT matrices have been updated by prior measurements, some values might not be relevant when a new placement is to be chosen, due to variations of the network QoS parameters. There is therefore a probability that the PSO heuristic, based on these biased values, could choose a placement resulting in a higher response-time than the previous placement.

4.5 Evaluation of a Reactive Adaptation Strategy

We designed a new adaptation strategy to overcome the problem found in the proactive strategy. This strategy enables the system to trigger a new deployment of the microservices in the case of a deterioration of the response-time of the application. At each iteration, the system evaluates the global response time of the application and the strategy compares this value with a threshold, corresponding to the worst



(a) Proactive strategy

(b) Reactive strategy

Figure 4: Variations of the application response-time with different adaptation strategies.

acceptable response-time. We defined this threshold from prior experiments as the average of the 10 last global response-times multipled by 110%. If the current global response-time for the application exceeds this threshold, the strategy will trigger the selection of a new placement of the microservices. We have chosen to keep in this strategy the principle of a periodic adaptation as a mean to continuously update the values inside the RT matrices.

This strategy will be able to mitigate at least two situations of response-time deviation. The first situation occurs when variations of the network QoS parameters have a significant impact on the response-time. In such case, the values inside the RT matrices should indicate which microservices are responsible for the deviation of the response-time of the application. Based on these values, the PSO heuristic will disqualify these particular locations when selecting a new placement. Another situation is the case when a periodic adaptation results in a deterioration of the reponse-time. The updated values in the RT matrices should prevent PSO heuristic to choose a poor solution for this new adaptation.

We evaluated this reactive strategy on our test infrastructure using the same methodology as the previous strategy evaluation. Figure 4 (b) shows the variations of the response-time of the application for one of the most representative experiment. Periodic adaptations of the placement are represented in this graph using vertical plain lines. Adaptations triggered by the reactive strategy are represented using vertical dashed lines. We observed that most of adaptations due to the reactive strategy occurred right after a periodic adaptation which resulted to a poor placement solution. We also observed reactive adaptations due to a deviation of the response-time corresponding to variation of network QoS parameters.

In some cases, we noticed that the strategy triggers repeated adaptations. During these iterations, the placement proposed after a reactive adaptation request did not result in an improvement of the response-time. The strategy will therefore trigger a new adaptation until an improvement can be observed. To prevent an infinite-loop, we limited the number of consecutive adaptations the strategy can trigger. As a positive side effect, these tests of alternative placements enable the system to update others values inside the RT matrices.

We performed several experiments to get a representative view of the behavior of both strategies with different network QoS variations. Table 3 presents the key indicators extracted from these experiments. This table shows that the reactive strategy presents a higher ratio of adaptations resulting in an improvement of the response-time than the proactive strategy. It can be noted that the strategy can still make poor choices. Indeed, even if the RT matrices are updated more frequently, they still can present some inconsistencies with current response-times.

Table 3: Key indicators for evaluated strategies.

Strategies	Proactive	Reactive
Elapsed Time (hours)	248	180
Adaptations	81	83
RT improvement $> 10\%$	30	45
RT deterioration $> 10\%$	27	16
Stable RT	24	21

5 CONCLUSION AND FUTURE WORKS

In this article we have considered the case of a microservice-based application deployed on devices distributed on the edge of the network. We observed that variations of parameters on these edge devices, such as network QoS, may impact the response-time of the application. We proposed a solution to dynamically adapt the placement of the microservices to these variations. The adaptation process we described in this article is based on the knowledge of the response-time of each microservice. These values are directly measured by the devices participating to the application and stored in RT matrices. The PSO heuristic uses these values to propose adapted placement solutions.

We evaluated on a production infrastructure two strategies to decide when the system should adapt the placement of the microservices. A first proactive strategy tries to periodicaly adapt this placement. We observed that half of the adaptations resulted in a deterioration of the response-time. We explained that, when such adaptation occurred, the RT matrices were inconsistent with the actual response-times. We proposed to supplement this strategy with a reactive behavior which allows the system to trigger a new adaptation in case of deviations of the response-time. We concluded from our experiments that this strategy gave satisfactory results.

We observed that the reactive strategy resulted in consecutives iterations where several placements are tested. These iterations led to a perceptible unstability of the response-time of the application. To keep a stable experience for the users, we propose, as an improvement of the solution, to test these alternative placements in parallel with the current deployment of the application. Thanks to the flexibility of microservices, a subset of the clients can transparently use an alternative deployment of the application. Measurements from the devices participating in this deployment will help the heuristic to find a suitable placement solution.

REFERENCES

Aryal, R. G. and Altmann, J. (2018). Dynamic application deployment in federations of clouds and edge resources using a multiobjective optimization AI algorithm. In *Third International Conference on Fog and Mobile Edge Computing (FMEC)*.

Aschoff, R. and Zisman, A. (2011). QoS-Driven Proactive Adaptation of Service Composition. In *Service-Oriented Computing*, Springer.

Brogi, A., Forti, S., Guerrero, C., and Lera, I. (2019). How to Place Your Apps in the Fog – State of the Art and Open Challenges. *Software: Practice and Experience*.

Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Lo Presti, F., and Mirandola, R. (2017). MOSES: A Platform for Experimenting with QoS-Driven Self-Adaptation Policies for Service Oriented Systems. In *Soft. Eng. for Self-Adaptive Systems*. Springer.

Cunsolo, V. D., Distefano, S., Puliafito, A., and Scarpa, M. (2009). Volunteer Computing and Desktop Cloud: The Cloud@Home Paradigm. In *IEEE International Symposium on Network Computing and Applications*.

Egger, S., Reichl, P., Hoßfeld, T., and Schatz, R. (2012). Time is bandwidth ? Narrowing the gap between subjective time perception and Quality of Experience. In *IEEE International Conference on Communications*.

Kephart, J. and Walsh, W. (2003). An architectural blueprint for autonomic computing. Technical report, IBM.

Khan, A. M., Selimi, M., and Freitag, F. (2014). Towards Distributed Architecture for Collaborative Cloud Services in Community Networks. In *6th International Conference on Intelligent Networking and Collaborative Systems (INCoS'14). Salerno, Italy: IEEE.*

Khorsand, R., Ghobaei-Arani, M., and Ramezanpour, M. (2018). FAHP approach for autonomic resource provisioning of multitier applications in cloud computing environments. *Software: Practice and Experience*.

Mutanu, L. and Kotonya, G. (2018). What, Where, When, How and Right of Runtime Adaptation in Service-Oriented Systems. In *Service-Oriented Computing – ICSOC 2017 Workshops*, Lecture Notes in Computer Science, pages 30–42. Springer.

Nishio, T., Shinkuma, R., Takahashi, T., and Mandayam, N. B. (2013). Service-oriented Heterogeneous Resource Sharing for Optimizing Service Latency in Mobile Cloud. In *First International Workshop on Mobile Cloud Computing & Networking*.

Singh, S. and Chana, I. (2015). QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review. *ACM Comput. Surv.*, 48.

Skarlat, O., Karagiannis, V., Rausch, T., Bachmann, K., and Schulte, S. (2018). A Framework for Optimization, Service Placement, and Runtime Operation in the Fog. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 164–173.

Stevant, B., Pazat, J.-L., and Blanc, A. (2018). Optimizing the Performance of a Microservice-Based Application Deployed on User-Provided Devices. In *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 133–140.

Yousefpour, A., Patil, A., Ishigaki, G., Kim, I., Wang, X., Cankaya, H. C., Zhang, Q., Xie, W., and Jue, J. P. (2019). FOGPLAN: A Lightweight QoS-Aware Dynamic Fog Service Provisioning Framework. *IEEE Internet of Things Journal*, 6(3):5080–5096.

Zhang, W., Hu, Y., Zhang, Y., and Raychaudhuri, D. (2016). SEGUE: Quality of Service Aware Edge Cloud Service Migration. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 344–351.