Optimizing the performance of a microservice-based application deployed on user-provided devices

Bruno Stévant
IMT Atlantique, IRISA
bruno.stevant@imt-atlantique.fr

Jean-Louis Pazat
Univ Rennes, Inria, CNRS, IRISA
jl.pazat@irisa.fr

Alberto Blanc
IMT Atlantique, IRISA
alberto.blanc@imt-atlantique.fr

***Abstract*—As an alternative to public clouds, devices on the user premises now have enough resources to host rich services that can even be accessed remotely. A community of users can leverage the devices provided by its members to build a large and highly-available system, comparable to a data-center. An application based a microservice-oriented design can be distributed over such system by deploying each service on different devices. The performance of application is however highly dependent of the CPU and network resources available on the devices hosting the services. Finding the placement that results in the optimal performance of the application is a NP-hard problem. We propose in this article a model of the application performance based on the response time of the microservices deployed on these devices. Based on this model, we propose a heuristic based on Particle Swarm Optimization in order to find optimal placement solutions in constant time. We tested the result of this heuristic on a platform emulating a network of edge devices with QoS parameters similar to residential access networks. We validated that the application shows good performance even if the network QoS is fluctuating.**

***Index Terms*—Edge computing, Performance modelization, QoS-aware placement, Particle Swarm Optimization, Microservice application design**

I. Introduction

Over the last decade, the number of public cloud providers has increased significantly. It is now possible to rent resources located in one or more data-center. Such an infrastructure can satisfy the needs of a large number of users, but it does have the downside that data must be moved (and often stored) in these shared data-centers, raising doubts about the privacy of the data and, potentially, its ownership (for instance, in the case of providers going out of business).

Data-centers are not alone in taking advantage of technological progress: user devices, including personal computer and set-top boxes, have ever increasing computing and storage capabilities. Computer networks offer higher data rates to the wider public, with the spread of Fiber To The Home (FTTH) solutions and high speed wireless networks. Thank to all this, it is conceivable for the devices hosted on the user premises to offer services to their owners, even remotely. Such a solution has the added advantage of addressing the data privacy and ownership issues mentioned above.

An obvious shortcoming of such an approach is that it introduces two single point of failures, namely the user device and the corresponding network connection. Either one can fail, rendering the service unavailable remotely. While modern hardware is reasonably reliable, such a solution is no match for the public cloud providers, with their redundant infrastructure, which allow them to offer highly available services.

A possible solution to overcome these limitations is for several users to cooperate, by pooling their individual resources to form a larger (redundant) system. A few devices (for instance between 10 and 20) can already be enough to significantly increase the availability and, potentially, the performance with respect to a single device. As an example, consider a sport club, whose member would like to share pictures and other material related to the club activities. One member decides to use the storage space available on one of his devices connected at home to let other members store and retrieve their photos. Comparing this solution with the service provided by a public platform raises two limitations. First, as these devices are provisioned for individual usage, they might not have enough local resources (CPU, storage, network) for a larger number of users. In our example, the storage space and the residential network will become bottlenecks as members of the club are uploading and browsing more photos. The second limitation comes from the lower availability of these on-premise devices. The photo-sharing storage will be unavailable anytime the device is powered off or experiencing a network outage. If multiple members of the sport club are willing to share their devices and network connectivity, they could significantly improve the situation, provided they have a large enough variety of network providers and usage patterns. Extended functionalities like generating thumbnails and presenting them in a web gallery can be offered to the community as the different devices contribute to the application with their computing and networking resources.

Members of the community will fully benefit from the cooperation between the devices as the application can be distributed over these devices. Many frameworks for distributing applications over user-provided devices have been already proposed, such as Cloud@Home[6], Nebulas[4], Community Cloud[12] or CNMC[16]. These solutions aim at creating an Infrastructure-as-a-Service (IaaS) interface on top of the user-provided devices where applications embedded inside virtual machines can be deployed and managed the same way as in a data-center. Such solutions however tends to be complex as they have to adapt to devices and networks which are much more heterogeneous and with a lower availability rate than the infrastructure of a data-center.

In this paper we focus on a higher level solution where the application is based on a microservice architecture[10]. In such an architecture, each basic functionnality of the application is provided by a separate microservice. The application can be considered as a composition of these different microservices. Distributing the application on the user-provided devices is then possible by deploying each microservice on the different devices. Once each microservice is deployed and accessible from any other devices, the composition of these services offers a fully functional application.

Such distribution solution is very flexible as many possible placements of the services of the application are possible. The performance of application is however highly dependent of the chosen placement. Each service may indeed be deployed on devices with CPU and network resources that are insufficient for the task handled by the service. Finding the devices where to deploy the microservices which result in the optimal performance of the application is a difficult problem. The choice of a placement of the service has to match the requirements of the services and the resources available on the devices.

As a first contribution of this paper, we describe in section III a model to evaluate the performance of the application depending on the capabilities of devices chosen for the placement of the microservices. In section IV we define the problem of finding the placement of the services that optimize the performance as defined by our model. The Particle Swarm heuristic is used to find acceptable solutions to this problem in constant time. A second contribution of this paper is the evaluation of the results of this heuristic on a platform emulating user-provided devices interconnected by a network with similar capabilities to residential networks. We demonstrate in section V that with relatively little information on the infrastructure and on the application, the heuristic can find good solutions close to 15% of the optimal performance.

II. A Concrete Example: a Photo Sharing Application

In this paper, we consider a photo sharing application as the application used by the members of a community. Users of the application can upload their photos and browse a web gallery with thumbnails of the stored photos. This application has been designed as a composition of four microservices implementing the functionnalities of the application. Figure 1 (a) shows the microservices of the photo sharing application and their interactions:

- **WebUI** (UI): This service provides a web photo gallery with thumbnails and a form to upload photos.
- **PhotoHub** (PH): This service is used to store and retrieve the photo files.
- **ThumbHub** (TH): This service produces the thumbnail of a photo by resizing the original photo.
- **MetaHub** (MH): This service is used to stores and retrieve the photo metadata.

It is important to note that the microservices are inter-dependent. Even if each microservice handle a basic task, it may need to call other services to get some data or perform other tasks. In our case, the ThumbHub service relies on the PhotoHub service to get the original bitmap of the photo and on the MetaHub service for the metadata of the photo.

We want to distribute these microservices on different user-provided devices that present heterogeneous CPU and network capabilities. Figure 1 (b) shows a representation of these devices interconnected thanks to the Internet. Each device makes available a different amount of CPU resources depending on its hardware profile. The residential access network that connect the device to the Internet also has heterogeneous capabilities in term of bandwidth and lantency, depending on the technology used by the access network (DSL or Fiber).

Figure 1 (c) gives an example of a deployment of the application over these devices. Some devices are selected to host a service of the application, for example using a placement algorithm we describe later. Once the services are deployed, an external registry service, not described in this paper, is used to make the deployed services known and reachable from the other services. The services, being able to communicate with the each other, are then capable of handling the tasks of the application.

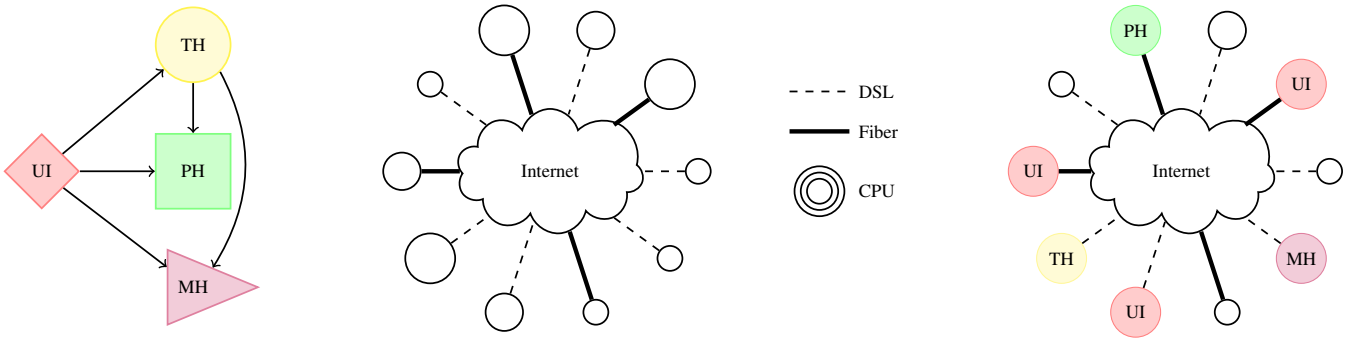
By deploying the microservices on the different devices, we distribute the tasks of the application. Many distributions are possible, each of them however results in a mixed performance for the application. The time for a service to handle a request is indeed dependent of the computing capabilities of the device, especially when this task requires some computations. For example, the ThumbHub service, which has to generate thumbnails by resizing an photo, will perform better if located on a device with large CPU resources.

As the microservices are communicating through a wide-area network, the time spent in forwarding the requests and replies over the network is highly impacting the performance of the application. In our example the performance of the photo-sharing application is depending on the available bandwidth for the PhotoHub service used to store and retrieve large photo files.

The global performance of the application is therefore highly dependent on the devices chosen for the placement of each microservice. Finding the devices where to deploy the microservices resulting in the optimal performance for the application is a difficult problem. The choice of a placement should consider the requirements of each microservice and the communication implied by their composition inside the application. It should match these requirements to the capabilities of the devices and of the network in order to optimally place the microservices where they will globally provide the best performance for the application.

III. Performance metrics

In this section we present a model for the performance of the application depending of the placement of its microservices. This model defines an objective function that evaluate the global performance of the application based on the response time of its services called from the differents clients of the application. Our model is based on the knowledge of the



(a) Microservices-based application (b) User-provided devices with different capabilities (c) Deployment of microservices on devices

Fig. 1: Deployement of an application based on microservices over user-provided devices.

response time for each microservices deployed on any device and requested from any other device.

The performance evaluated from the client side, which can be related to the Quality of Experience (QoE), is directly related to the response time of the application. In [15] the response time of the application is considered as a single metric to evaluate the possible gain from outsourcing part of the application to other devices with heterogeneous capabilities. In our case, we use the response time to evaluate the performance of the application in different deployment scenarios. Each client will evaluate the performance of the application as the sum of the response times of the calls to the application. In the case of the photo-sharing application, the different calls are: browsing the gallery of photos, downloading a full-size photo, downloading a thumbnail for a photo and uploading one photo.

Based on the values of the response times for the different services, we can evaluate the response time of the application as the sum of the response times for each microservices involved in the application. The objective function can be mathematically expressed as function f defined as follow:

- Let N be the set of devices in the infrastructure
- Let C be the set of devices used as clients to the application; $C \subset N$
- Let S be the set of services composing the application
- Let $D(s_i, s_j)$ be the dependency between each service s_i and s_j ; $D(s_i, s_j) = 1$ if s_i needs to call s_j, otherwise $D(s_i, s_j) = 0$
- Let $P(s)$ be the chosen placement for service s; $s \in S, P(s) \in N$
- Let $R_s(n_i, n_j)$ be the response time of a service s requested by device n_i when running on device n_j.

The objective function for one user is the response time for this user requesting the front-end service s_0 from its own node c. For the sake of simplicity, we consider that one service requests the other needed services sequentially. In this case, the response time is the sum of the response time of the others required services seen from the front-end s_0. These others services may also call other services, which imply the objective function to be computed recursively.

$$f(c, s_0) = R_{s_0}(c, P(s_0)) + \sum\nolimits_{s_k \in S} D(s_0, s_k) \times f(P(s_0), s_k)$$

The global objective function is the weighted sum of the objective function for each user. Let $W(c)$ the weight applied for user c:

$$f(s_0) = \sum\nolimits_{c \in C} W(c) f(c, s_0)$$

In the case of the photo-sharing application the set of services is:

$$S = \{s_0, s_1, s_2, s_3, s_4\}$$

where s_0 is the UI service, s_1 is the MH service, s_2 the PH service used to download a photo, s_3 the PH service used to upload a photo and s_4 the TH service.

The dependency D is built from the call graph of the application, depicted in figure 1 (a):

$$\begin{aligned} &D(s_0, s_1) = 1 && \\ &D(s_0, s_2) = 1 && D(s_4, s_1) = 1 \\ &D(s_0, s_3) = 1 && D(s_4, s_2) = 1 \\ &D(s_0, s_4) = 1 && \end{aligned} \tag{1}$$

These values show that the UI service (s_0) calls each of the other services and the TH service (s_4) calls services MH (s_1) and PH (s_2).

The objective function for one user can be expressed as :

$$\begin{aligned} f(c, s_0) = {} & R_{s_0}(c, P(s_0)) + R_{s_1}(P(s_0), P(s_1)) \\ & + R_{s_2}(P(s_0), P(s_2)) + R_{s_3}(P(s_0), P(s_3)) \\ & + R_{s_1}(P(s_4), P(s_1)) + R_{s_2}(P(s_4), P(s_2)) \end{aligned} \tag{2}$$

IV. Optimizing of the performance

In this section we discussed of the problem of optimizing the performance of the application based on the model introduced in the previous section.

The objective function 2 gives an evaluation of the response time of the application for one client from the placement $P(s)$ for each microservice s and the response time $R_s(n_i, n_j)$ for this service between two devices n_i, n_j. The placement

$P_{opt}(s)$ which results in the optimal performance of the application should minimize the objective function $f(s_0)$. Finding $P_{opt}(s)$ requires to find the values of $P(s_i) \in N;\ s_i \in S$ which minimize the sum presented in function 2. The number of possible placements is equal to $|N|^{|S|}$. The complexity of an exhaustive search for the optimal placement $P_{opt}(s)$ is therefore polynomial to the number of devices and exponential to the number of services.

As this problem is NP-hard, it is necessary to use a heuristic in order to find acceptable solutions in constant or linear times. We found that nature-inspired metaheuristics have shown good results for the problem of Quality of Service (QoS)-aware service composition [11], to which our problem can be considered as equivalent. Heuristics based on Genetic Algorithms or Ants-colony optimization are indeed particulary efficient for problems with a very large solution space as they can find acceptable solution in constant time. We selected Particle Swarm Optimization (PSO) [7] as a candidate for a metaheuristic to be applied to our problem. Such algorithm has already been used in [14] to solve the problem of choosing a composition of services based on QoS parameters.

By applying the PSO metaheuristic, we define a particle location as a placement $P(s_i) \in N;\ s_i \in S$. At the first iteration, the PSO algorithm initializes a swarm of k particles by randomly selecting their location, e.g choosing a random placement for the services. The objective function 2 is evaluated for each of these placements and the result is assigned to the corresponding particle as its current score. At each iteration, the PSO algorithm defines a new generation of the swarm where each particle is updated with a new location. The new location for a particle is computed from parameters such as its previous location, its velocity and the minimal score found in the previous iteration. By this mean, the particules will converge at each iteration to the particle with the current minimum score. Other local minima may also be found as new placements are evaluated. The algorithm may stop either when all particles have converged to the same minimum score or when it reaches a maximum number of iterations fixed as a parameter to the algorithm.

In order to validate the choice for this heuristic, we tested the PSO algorithm by using as input synthetic values for the response times of the service. We generated these values using a simple model described below. We then compare the score of the placement found by the heuristic to the score of the optimal placement found by exhaustive search. For the heuristic to be considered as valid, the score of the placement found by PSO should be relatively close to the optimal score compared to the score evaluated for others possible placements. The scalability of the heuristic is also evaluated by checking if the the minimal score found by PSO does not deviate from the optimal solution when the number of devices increases.

We generate the inputs for this test using a simple model for the response times of the service running on the different devices, e.g. the values for $R_s(n_i, n_j);\ n_i, n_j \in N$. For one pair (n_i, n_j), the value of $R_s(n_i, n_j)$ is chosen among values following a Pareto distribution whose parameters depend on the nature of the service s and of the interconnecting network between the devices n_i and n_j. In order to add some diversity in these values we use in this model two types of networks technology with different QoS parameters: Fiber (high symetrical bandwidth, low latency) and DSL (low assymetrical bandwidth, high latency). The Table I gives the parameters of the Pareto distributions (min value / average value) used to generate the response time $R_s(n_i, n_j)$ of each service s depending of the type of network connecting the devices calling the service (n_i) and the device hosting the service (n_j). For the test we choose that 50% of the devices use a DSL network and 50% use a Fiber network.

Service s	n_i \ n_j	DSL	Fiber
MetaHub	DSL	50/200	30/100
	Fiber	200/800	15/70
PhotoHub (get photo)	DSL	200/800	50/200
	Fiber	200/800	30/100
PhotoHub (put photo)	DSL	200/800	200/800
	Fiber	50/200	30/100
ThumbHub	DSL	100/300	40/150
	Fiber	100/300	40/150

TABLE I: Min/Average values for the response times (in ms) for the different services depending on the network used by client n_i and server n_j

We initialize the algorithm with the following parameters:

- Number of devices: $|N| \in \{10, 20, 30, \ldots, 100\}$
- Number of clients: $|C| = 0.2 * |N|$
- Weight applied to clients: $W(c) = 1\ \forall c$
- Number of particles: 100
- Max. number of iterations: 100

In addition to these parameters, we introduce some constraints to the solution to be found by the PSO heuristic. A first constraint is related to the PhotoHub service that is represented in our model by two virtual services : download an existing photo file and upload a new photo file. These virtual services are defined in our model as s_2 and s_3. As these services are offered by the same microservice, they should be deployed in the same device. The first constraint for the heuristic is therefore $P(s_2) = P(s_3)$. A second constraint is defined to avoid trivial solutions where all services are deployed on the same device. Such solutions give good results in our model but are not realistic as the concurrency between the services reduces the performance. To reduce concurrency, all services should be deployed on different devices, except for s_3 which is colocated with s_2. The second constraint applied to PSO solutions is $P(s_i) \neq P(s_j)\ \forall i, j \in \{1, 2, 4\}$

After providing these parameters and constraints, the PSO algorithm is able to produce a result which is the placement evaluated with the minimal score. It is important to note that several executions of the algorithm with the same parameters may produce different results. It can be explained as the particles may not have converged before the algorithm reaches the maximum number of 100 iterations we configured. By increasing the number of iterations, the algorithm produces more stable results. However it also increase the execution

time. We established in our case that 100 iterations is a reasonable compromise for the PSO algorithm to produce consistent results.

We tested the PSO algorithm in different test cases by increasing the number of devices from 10 to 100. The execution time of the PSO algorithm was constant as the number of iterations is limited and the evaluation of the objective function does not depend on the number of devices. The execution time was below 1 second on a 3GHz 4-core server. We also performed an exhausive search of the optimal solution by evaluating the score of all possible deployments. The execution time of this search dramatically increased with the number of devices, as the complexity of the problem, when considering this parameter, is polynomial.

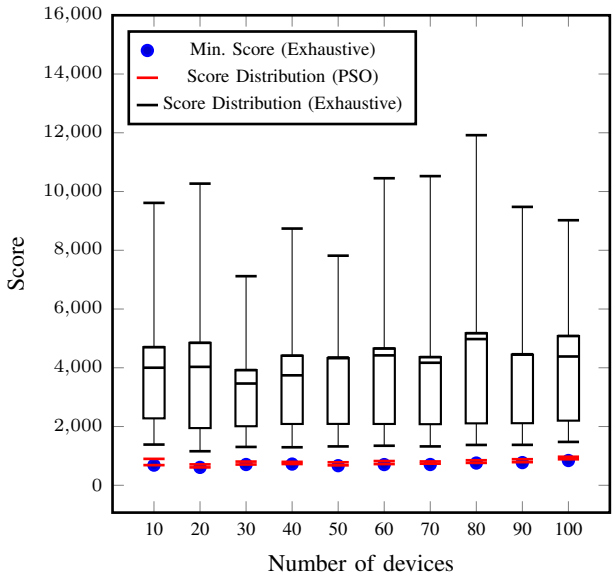


Fig. 2: Results of PSO heuristics compared to exhaustive search.

Figure 2 shows the results of both PSO and exhaustive search algorithms for the different test cases. The score of the optimal solution for each test case found with exhaustive search is marked with a blue dot. The two red horizontal lines limit the score interval of 90% of the solutions found by 50 executions of the PSO algorithm. The boxes show the distribution (5-percentile, 25-percentile, average, 75-percentile, 95-percentile) of the scores found with exhaustive search.

From this results we can conclude that the PSO heuristic is able to find placements for which the performance for the application is very close to the optimal solution found with exhaustive search. These placements are included in the 5% of the possible placements with the lowest score. Moreover the PSO heuristic is able to find such placement in a maximum of 100 iterations.

The code used for this test is written in Python[1] and uses the `pyswarm` library as implementation of the PSO algorithm.

[1]Code available at https://github.com/bstevant/eval-pso-photosharing

V. Test an validation on an emulated network

We describe in the previous section the test used to validate the PSO algorithm as a heuristic to find placements of microservices optimizing the performance of the photo-sharing application. In this test, the response times of the services $R_s(n_i, n_j)$ are static values generated using a simple model and the objective function use these values to evaluate each placement. But in real-life systems, the measured response time of a service is a random variable depending on the device and the network usage.

In order to evaluate the impact if such variations on the objective function defined in our model, we use a network emulation platform reproducing the QoS parameters of different technologies for residential access networks (DSL, Fiber) and the variations of the network performance (jitter, packet loss). By measuring the response time of actual microservices connected to such network emulation, we can evaluate in realistic conditions the performance of the application defined by the objective function.

We selected Containernet[2] as the network emulation plateform for this evaluation. Containernet is a fork of the well-known Mininet network emulator, which can connect Docker containers to the emulated network. It also provide the functionnality to define, for each link of the emulated network, the QoS parameters such as bandwidth, latency, jitter and packet loss.

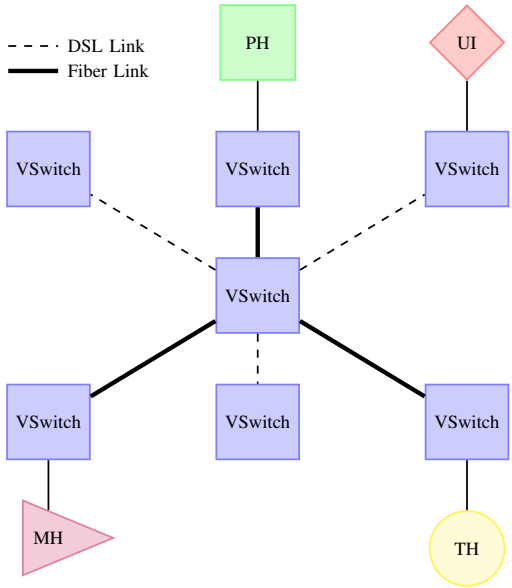


Fig. 3: Example of a topology used in the network emulator for evaluating a deployment of microservices over devices on user-premise.

Containernet allows us to define network topologies emulating devices on user-premise interconnected with different network technologies. An example of such topology is shown in Figure 3. Each user-provided device is emulated by a virtual

[2]Containernet website: https://containernet.github.io/

Technology	Uplink BW	Downlink BW	Latency
Fiber	100Mb/s	100Mb/s	5ms
DSL	2Mb/s	20Mb/s	25ms

TABLE II: QoS parameters for emulation of residential access networks.

switch to which containers of the microservice assigned to this device are connected. Each virtual switch representing a device is then attached to a central switch with a link emulating the residential access network. This link is configured with the QoS parameters of the network technology (DSL or Fiber) chosen for the device. The QoS parameters defined for each type of link are listed in Table II

Using this platform for network emulation, we are able to define different test cases by specifying the number of devices and the type of network for each device. Once the topology defined for the use case is instanciated in the network emulator, the actual containers embedding the microservices of the photo-sharing application are attached to the emulated network. Each container is connected to the switch (i.e. the device) specified in the placement of the service. Once all the microservices are attached, it is possible to measure the performance of the application in realistic conditions.

At a first step, we need to bootstrap the values for $R_s(n_i, n_j);\ n_i, n_j \in N;\ s \in S$ used by the objective function used by PSO to evaluate a placement. Two methods can be applied to bootstrap these values. It is possible to measure them exhaustively by deploying the service on each device and measure its response time from the other devices. This process is cumbersome and can be relatively expensive when the number of devices is important. Another solution is to estimate the value for the response time based on the QoS parameters of the platform. We made sample measurements of the response time of the service on different devices and we infered the response time for the other devices by linear regression on the bandwidth and latency parameters.

In addition to the values for $R_s(n_i, n_j)$ infered using the method described above, we defined the following parameters for the PSO algorithm:

- Number of devices: $|N| = 50$
- Number of clients: $|C| = 10$
- Weight applied to clients: $W(c) = 1\ \forall c$
- Number of particles: 100
- Max. number of iterations: 100
- Constraints: $P(s_2) = P(s_3); P(s_i) \neq P(s_j)\ \forall i, j \in \{1, 2, 4\}$

Configured with these parameters, the PSO heuristic is able to produce a placement for the microservices of the photo-sharing application. The containers embedding these services are deployed on the virtual devices specified by the placement. The real performance of the application is then measured from the clients as the sum of the response times of the calls to the application.

To evaluate the stability of the system, we introduce some variations on the performance of the network by adding a jitter parameter. In wide-area network the measured delay shows variations as a packet is forwarded in the network among many other packets. In order to reproduce this phenomenon, the network emulator use the well-known `netem`[3] module to manage the delay variation. We define the jitter as a percentage of the nominal latency for the link. The value for the delay applied to a packet on a link is the sum of the nominal latency and a random value from 0 to $jitter * latency$, following a normal distribution.

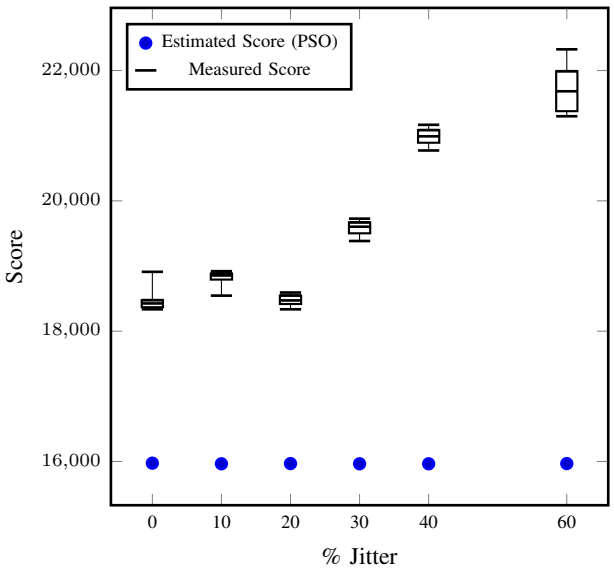


Fig. 4: PSO evaluated score compared to measured performance on a emulated network with different jitter values.

We tested the PSO algorithm on a single test case (same topology, same devices used as clients) with jitter increasing from 0% to 40% of the nominal latency for the link. The performance of the application is measured by the clients by requesting the different services of the application 10 times each. The values of the optimal performance estimated by PSO and the global performance measured by all clients are shown in Figure 4. These measurements show that for a jitter below 20% of the nominal latency of the link, the performance measured for the application is stable at approximatively 15% of the optimal performance. For jitter above 20%, the measured response times increase and deviate from the optimal performance.

We can conclude from this measurement that the performance of the photo sharing application, as defined in our model, is not impacted by the fluctuation of the network. With a jitter below 20% of the nominal latency, the measured performance of the application stays in the same interval of 15% of the optimal performance estimated by PSO. In such condition, we can be confident that the PSO heuristic is able to select an acceptable solution for the placement.

[3] `netem` module: https://wiki.linuxfoundation.org/networking/netem

VI. Related works

The contribution of this article is connected to various areas of research where the problematic the performance of applications distributed on edge devices have has been studied.

Volunteer computing and Grid computing are well studied areas of research with many contributions. Some of them are focused on performance considerations such as [3] and [8]. In such context, the performance is measured as task throughput. Job scheduling strategies are defined by considering resources availability and reliability. They aim at optimizing the throughput by reducing the idle time at the device level. The optimization target is different in our case as we are focused on application responsiveness.

In a service-oriented approach, a distributed application can be considered as a composition of service executing a workflow. Choosing a service composition that optimizes the global QoS of the workflow is a problem considered by many contributions[1]. Some proposed solutions to this problem are based on Linear or Mixed Integer Programming [19][2]. Other heuristics to this problem have been proposed such as Genetic Algorithm[13] or Particle Swarm Optimization[14]. The problem of service composition and workflow scheduling is however slightly different from our problem. In such case the services locations are given as input to the scheduling strategy that should decide which service instance to used in the workflow. In our case the services can be located any devices, which increase the complexity of the problem.

The placement of services considering devices with heterogeneous CPU and network capabilities also has been studied in various articles. In [9] the placement is considered in the case of servers connected by an overlay network over a wide area network whereas in [17] the servers are located in different public clouds. Each article proposes a heuristic to find an optimal placement by using a fine-grain model of the performance considering the network QoS parameters and the CPU resources. In [16] the authors present the use case of hosting the service in edge micro-clouds. This article proposes as a heuristic to select in the model the significant parameters according to the requirement of the services (e.g. bandwidth demanding or latency sensitive services). In our approach we adopt a higher level model by the use of the response time as the single metric for the evaluation of the performance.

Fog computing[18] is an emerging area of research that focus in task migration from the cloud to the edge. In [5], the use case of deploying a game service on edge servers to optimize the player latency is presented. These edge servers are selected in a decentralized way by a voting based protocol. This approach is interesting as our centralized approach may have some limitations if the number of devices participating in the system may increase. Our problem is yet more complex as we consider the placement of inter-dependent services, while the game service is standalone.

VII. Conclusion and Future works

In this article, we presented the use case of an application defined as a composition of microservices which are distributed on several user-provided devices. An adapted placement of the microservices is crucial for the performance of the application. By proposing a model of the performance based on the response time of the services, we showed that finding a placement resulting in optimal performance is a NP-hard problem. In order to find acceptable solutions to this problem we proposed a heuristic based on Particle Swarm Optimization. We demonstrated by simulation that such heuristic gives satisfactory results in constant time. In order to validate the heuristic in realistic conditions, we evaluate the performance of the application on an emulated network. When deployed following the placement selected by the heuristics, the application show stable performance below 20% of jitter. We conclude that even if the PSO heuristic use static infered values as the response times of the services, the selected placement still results in acceptable performance for the application even if the network QoS is fluctuating.

In addition to the response time of the services, the client behavior is another key parameter in our model for the performance evaluation of the application. The existing model consider that all clients are requesting the service of the application with equal probability, and that each client make the same number of requests. Some parameters of the model can be adjusted to approach the average behavior of the client. It is nevertheless important to evaluate how fluctuations in the request distribution may impact the performance of the application. The network emulation platform also gives the opportunity to emulate concurrent requests from different clients to the services. As our model considers an unloaded network, it should be extended to avoid concurrency. One solution is to allow one service to define multiple instances that share the load of processing the incoming requests. The placement should then decide where the different instances should be located.

References

[1] Ehab Nabiel Alkhanak, Sai Peck Lee, and Saif Ur Rehman Khan. Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities. 50:3–21.

[2] Mohammad Alrifai and Thomas Risse. Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 881–890. ACM.

[3] D. P. Anderson and G. Fedak. The Computational and Storage Potential of Volunteer Computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06*, volume 1, pages 73–80.

[4] Abhishek Chandra and Jon B. Weissman. Nebulas: Using Distributed Voluntary Resources to Build Clouds. In *HotCloud*.

[5] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. A hybrid edge-cloud architecture for reducing on-demand gaming latency. 20(5):503–519.

[6] V. D. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa. Volunteer Computing and Desktop Cloud: The Cloud@Home Paradigm. In *2009 Eighth IEEE International Symposium on Network Computing and Applications*, pages 134–139.

[7] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS '95*, pages 39–43.

[8] A. Essafi, D. Trystram, and Z. Zaidi. An Efficient Algorithm for Scheduling Jobs in Volunteer Computing Platforms. In *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 68–76.

[9] Jeroen Famaey, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. Network-aware service placement and selection algorithms on large-scale overlay networks. 34(15):1777–1787.

[10] Martin Fowler and James Lewis. Microservices.

[11] C. Jatoth, G. R. Gangadharan, and R. Buyya. Computational Intelligence based QoS-aware Web Service Composition: A Systematic Literature Review. PP(99):1–1.

[12] Amin M. Khan, Mennan Selimi, and Felix Freitag. Towards Distributed Architecture for Collaborative Cloud Services in Community Networks. In *6th International Conference on Intelligent Networking and Collaborative Systems (INCoS14). Salerno, Italy: IEEE*.

[13] A. Klein, F. Ishikawa, and S. Honiden. SanGA: A Self-Adaptive Network-Aware Approach to Service Composition. 7(3):452–464.

[14] Wenfeng Li, Ye Zhong, Xun Wang, and Yulian Cao. Resource virtualization and service selection in cloud logistics. 36(6):1696–1704.

[15] Takayuki Nishio, Ryoichi Shinkuma, Tatsuro Takahashi, and Narayan B. Mandayam. Service-oriented Heterogeneous Resource Sharing for Optimizing Service Latency in Mobile Cloud. In *Proceedings of the First International Workshop on Mobile Cloud Computing & Networking*, MobileCloud '13, pages 19–26. ACM.

[16] Mennan Selimi, Davide Vega, Felix Freitag, and Lus Veiga. Towards Network-Aware Service Placement in Community Network Micro-Clouds. In *Euro-Par 2016: Parallel Processing*, pages 376–388. Springer, Cham.

[17] Moritz Steiner, Bob Gaglianello Gaglianello, Vijay Gurbani, Volker Hilt, W.D. Roome, Michael Scharf, and Thomas Voith. Network-aware Service Placement in a Distributed Cloud Environment. 42(4):73–74.

[18] Luis M. Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. 44(5):27–32.

[19] Chen Wang. A QoS-Aware Middleware for Dynamic and Adaptive Service Execution.