Towards a participatory cloud infrastructure for hosting services: QoS-aware dynamic orchestration of microservices

Bruno Stévant, Jean-Louis Pazat, Alberto Blanc

Towards a participatory cloud infrastructure for hosting services: QoS-aware dynamic orchestration of microservices.

Bruno Stévant
IMT Atlantique, *IRISA*
Rennes, France
bruno.stevant@irisa.fr

Jean-Loui Pazat
INSA Rennes, *IRISA*
Rennes, France
jean-louis.pazat@irisa.fr

Alberto Blanc
IMT Atlantique, *IRISA*
Rennes, France
alberto.blanc@irisa.fr

***Abstract*—A Participatory Cloud Infrastructure (PCI) is a collaborative framework designed to host services for a virtual community thanks to residential devices contributed by its members. In this work, we deploy on these devices a photo-sharing application based on microservices. The placement of services on devices significantly influences the overall performance of the application. To address this concern, we present a heuristic able to find a placement that approximates the ideal application response-time in constant time. Our proposed approach has undergone validation within a real infrastructure, revealing the necessity for dynamic adjustments of service placements. Consequently, we introduce a comprehensive adaptation process for placement and an associated specialized strategy.**

***Index Terms*—participatory cloud infrastructure, microservices, performance optimization, dynamic placement adaptation**

I. Introduction

A. Participatory Cloud Infrastructure: definition and usage

Virtual communities on the Internet unite around online tools that facilitate interaction between members. These tools generate shared content from individual contributions. One exemple of such usage in a sports club's virtual community could be photo sharing and collaborative calendar editing. Community requirements for tool adoption include functional alignment with community needs and non-functional criteria, including universal accessibility, sustained availability, non-restrictive performance, acceptable financial implications, and assurance of data security and confidentiality.

The concept of Participatory Cloud Infrastructure (PCI) is an alternative to traditional hosting solution like service platforms or dedicated servers. It revolves around building a hosting solution from resources contributed by its members. Participants share computing or storage resources from devices they own at home (computers, home appliances, virtual machines). This sharing is realized through the installation of specific software on each device, ensuring the coordination of shared resources within the community to create a runtime environment for applications.

The PCI is essentially an extension of domestic self-hosting on a community scale, retaining properties such as data privacy and community control over tools and services. Unlike domestic self-hosting where the execution context is provided by a single device, the participatory infrastructure relies on multiple devices in different households. This approach enhances hosting solution availability, scalability and cost sharing between participants.

979-8-3503-7128-4/24/$31.00 ©2024 IEEE

B. Challenges

The proposed solution of a PCI raises several technical challenges. A first challenge involves deploying applications based on the available resources of the participative infrastructure. Coordinating devices and utilizing their resources for each application's execution requires a mechanism to automatically deploy applications on these devices and transfer their data. Considering the diverse profiles of the participating devices, the deployment system must account for storage capacity, processing power, and network quality of the residential access network.

A second challenge is to maintain an acceptable user experience for hosted applications despite the variable availability of resources. Since domestic network resources are concurrently used by household members for various purposes, the solution must not compromise these priority uses while ensuring a good user experience for the hosted applications. Adapting to the fluctuating availability of resources on each device and addressing potential equipment failures are crucial aspects of maintaining a high-quality user experience.

C. Existing approaches

We identified 3 differents approaches of PCIs that could meet our requirements : Voluntary clouds, Peer-to-Peer (P2P) clouds and Community clouds.

The proposed voluntary cloud solutions, such as Nebula [10], cuCloud [9], and Cloud@Home [2], leverage the concept of voluntary computing (as in BOINC) but with a cloud-centric approach, aiming to create a unified execution context from voluntarily contributed distributed resources. Nebula focuses on creating a computation infrastructure for intensive tasks, distinguishing between equipment for data storage and computation. cuCloud, designed for Desktop Cloud scenarios,

utilizes shared resources on site for hosting virtual machines, outperforming Amazon EC2 in latency. Cloud@Home aims to provide on-demand infrastructure services using voluntarily shared domestic equipment, implementing an architecture with dedicated brokers for resource management and an orchestrator for virtual machine deployment.

P2P cloud solutions, like P2PCS [1] and MycoCloud [3], aim to establish on-demand hosting infrastructure using decentralized resources without relying on centralized servers for resource management and application deployment. P2PCS focuses on creating a decentralized infrastructure from locally or geographically distributed resources, designed for applications benefitting from proximity to users like video streaming, online gaming, and collaborative applications. On the other hand, MycoCloud, building on the groundwork of MycoNet and MycoLoad, emphasizes a service-oriented approach within a large-scale P2P network.

Community cloud solutions leverage participatory networks, such as Wireless Community Networks, to provide application hosting services with a cloud-centric approach. The Cloudy prototype [7] aims to harness the computational resources of devices available on the Guifi.net community network in Spain by deploying various services as software containers on each device. Another study [11] expands Cloudy to distributed applications like video streaming and 3-tier web architecture, focusing on optimal component placement on devices. The proposed algorithm instantiates the application's component graph on the network connectivity graph, minimizing application response-time by optimizing link utilization in the community network.

D. Our proposition

The study of existing solutions guides the identification of key contributions to address challenges in realizing a participative hosting infrastructure. Infrastructure-as-a-Service solutions like Cloud@Home and CuCloud exhibit limitations, requiring substantial virtualization support at the equipment level. Task-centric approaches like Nebula demand significant adaptation for traditional collaborative applications. Service-oriented solutions like MycoCloud or Cloudy, tailored for deployment on home devices with varied and limited computational resources, have promising extensions to applications built on microservices.

Based on these conclusions, we chose, for our PCI design, a Platform-as-a-Service approach, facilitating microservices hosting. It incorporates mechanisms for deploying microservices on participating devices, encompassing data provisioning, microservice instantiation, networking across devices, and inter-microservice discovery. Our study focuses on maintaining an acceptable user experience of the services deployed on such platform.

II. Response Time management

This section introduces a model for application response-time based on placement and infrastructure parameters, and propose a heuristic we detailled in [12].

A. Use Case

This study investigates the implementation of a participatory hosting infrastructure for a community scenario involving 10 to 100 individuals. The community's objective is to facilitate photo sharing from events. To enable such usage, the community agreed on the deployment of a photo-sharing application, structured as microservices, to be hosted on a PCI built from domestic devices in members' households.

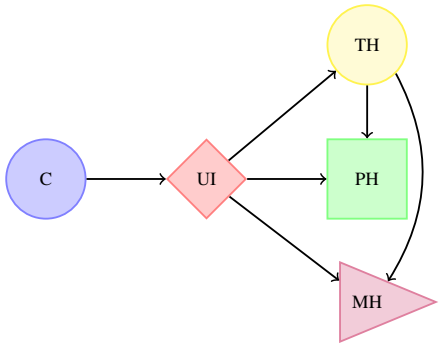

Fig. 1: Architecture of the photo-sharing application.

We chose, as target application to be deployed on this PCI, a photo-sharing web-service exemplifying microservices architecture principles. The application composed of four independent services with loose coupling - MetaHub (MH), PhotoHub (PH), ThumbHub (TH), and WebUI (UI) — handling metadata, binary data retrieval, thumbnail generation, and user interface, respectively. The microservices interact through defined interfaces, forming a dependency graph with constant interactions during application execution, as shown in Figure 1. The client (C) interacts directly with the UI service that serves as the front-end to the application.

In this architectural framework, microservices represent distinct functions of the application, and their interdependencies necessitate network communication. The process responsible to manage the response-time of the application should strategically select devices for hosting each microservice, considering device characteristics and service-specific requirements. Our study focuses on the decision-making process for optimal microservices placement to enhance overall application performance within the PCI.

B. Hypothesis

In this scenario, community members contribute their device resources for microservices hosting within a PCI. The model depicts an architecture comprising devices with varying computational capabilities interconnected through networks. Two device categories are considered based on computational capacities (low and high), as well as two connectivity types (Digital Subscriber Line (DSL) and Fiber To The Home (FTTH)). We assume in this study that the impact of storage type is negligible on the application response-time.

Considering requests to the application, we assume constant photo and thumbnail sizes, ensuring reproducible response-

times under stable resource usage conditions. Each microservice must be deployed at least once on the infrastructure, facilitated by a manager ensuring code and data availability. We limit the study to a scenario where each microservice is deployed only once, allowing flexibility in service placement without affecting application functionality.

C. Monitored indicators

To characterize the performance of an application deployment, we consider response-time as the primary indicator, which holds a significant value for user experience. The chosen response-time indicator is quantifiable and decomposable, measuring the interval between user request and application response. This decomposition into request transmission, processing, and response transmission intervals allows for effective optimization of the user experience. The application's response-time depends on various parameters, and our study aims to identify factors that can be influenced within the given context, particularly focusing on processing and data transmission times.

Processing time depends on the application's computational needs and processor characteristics, while transmission time relies on data quantity and network characteristics. The fixed nature of processing and data transfer requirements, along with processor and network capabilities in a PCI, necessitates strategic placement of the application to optimize response-time. The complexity increases for distributed, microservices-based applications, as each microservice has distinct computational and communication requirements.

D. Response time enhancement

1) Model for application response-time: We have to consider in our model the interdependent nature of the microservices. In terms of response-time, this implies that the observed latency for a request to a single microservice might depend of the latency of susequent requests originated by the microservice itself to others. In order to reduce this complexity, we introduce a metric called the specific response-time for a single microservice. This metric includes the data transmission time for both request and reply, and the local processing time of the request needed by the microservice.

This metric depends on the service itself, on the device where the service is deployed and on the device requesting the service. We can compile the different possible values for this metric in the form of a matrix $R_s(m, n)$ with s the considered service, m the requesting device and n the device hosting the service. We assume, in our model, that is possible to estimate the specific response-time of each microservice for each pair (m, n). We can then evaluate the response-time of the application as the sum of the specific response-times for each microservices involved in the application.

We need the following definitions in order to define the objective function f:

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

The response-time of the application for one client node c considering the placement P is measured as the response-time of the front-end service α from the node c. For the sake of simplicity, we consider that one service makes requests to the other services sequentially. In this case, the response-time of the application is the specific response-time of α added to the sum of the response-time of the others required services used by α. These others services may also call other services, implying that the objective function to be computed recursively. The response-time $t_{c,\alpha}(P)$ can be expressed as:

$$t_{c,\alpha}(P) = R_\alpha(c, P(\alpha)) + \sum\nolimits_{k \in S} D(\alpha, k) \cdot t_{P(\alpha),k}(P) \quad (1)$$

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

$$f(P) = \sum\nolimits_{c \in C} W(c) \cdot t_{c,\alpha}(P) \quad (2)$$

2) Heuristic: The placement P_{opt} that results in the optimal performance of the application is the one that minimizes the objective function $f(P)$. Finding P_{opt} requires finding the values of $P(s) \in N;\ s \in S$ which minimize the sum in the objective function 2. The number of possible placements is equal to $|N|^{|S|}$. As finding P_{opt} is an NP-hard problem, we consider using a heuristic in order to find acceptable solutions in constant or linear time. We found that nature-inspired metaheuristics, like Particle Swarm Optimization (PSO) [4], have shown good results for the problem of Quality of Service (QoS)-aware service composition [5] [8], to which our problem can be considered as equivalent.

We evaluated, on different use cases, the response-time of the placement found by the PSO heuristic and compared them to the minimum response-time produced with the optimal placement P_{opt}, which we found by exhaustive search. For the heuristic to be considered as valid, the response-time obtained with the placement found by PSO should be relatively close to the optimal response-time. We produced different use cases by simulating the response-time of the services with synthetic $R_s(n, m)$ values. We generated these values using a simple model taking as input the computing resource of node m hosting the service and the technology used as residential access network for each node n and m. The number of nodes and the distributions of computing resources and network technologies are the parameters we tuned to generate different use cases for our evaluation. We introduce an additional constraint to avoid trivial solutions where all services are deployed on the same device.

We tested the PSO algorithm in different 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 four-core server. We also performed an exhausive search for the optimal solution by evaluating the score of all possible deployments. Such exhaustive search might take up to 20 hours for largest use cases.

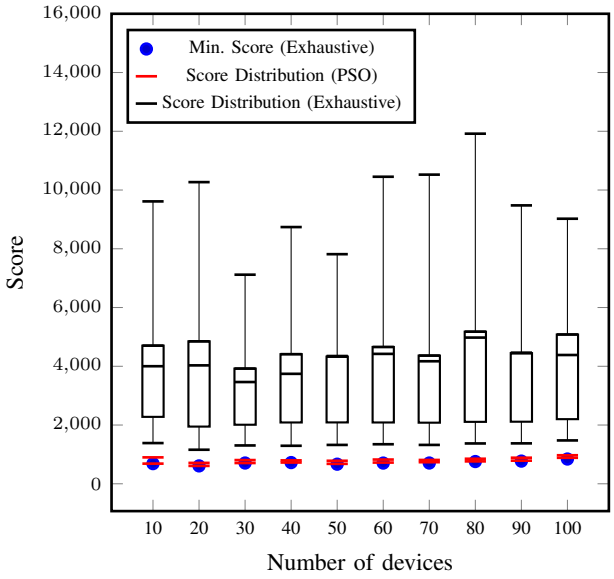

Fig. 2: Results of PSO heuristics compared to exhaustive search.

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 delimit the score interval containing 90% of the solutions found by 50 different executions of the PSO algorithm. The boxes show the distribution (5-percentile, 25-percentile, average, 75-percentile, 95-percentile) of the scores of all the possible placements (found with exhaustive search).

From these results we can conclude that the PSO heuristic is able to find placements for which the response-time for the application is very close to the optimal solution found with an 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.

III. Evaluation on real conditions

A. Test infrastructure

We deployed a test infrastructure using eight small-factor computers hosted by volunteers on their premises. We integrated in these devices the tools required for the remote deployement of the microservices and a framework enabling service discovery and communication. The deployment of the application is orchestrated by a remote server which manages the activation on designated devices of the different microservices of the application. In order to automate our tests, this server manages as well the devices designated as clients of the application, from which emulated requuests to the application are generated.

The physical devices are connected to the Internet through different types of residential networks. Six of these devices were connected through high-speed fiber networks that can achiever 100Mbit/s in each direction. Two devices were connected through a VDSL residential network which offers am asymmetrical bandwidth of 80Mbit/s downstream and 20Mbit/s upstream. The deployed devices were dedicated to our tests, therefore computing resources are always available to handle the workload of the application. However the network between each devices was shared with the users traffic on their residential network, then globally with other traffic on the Internet.

B. Evaluation methodology

Several experiments were conducted to evaluate the application's response-time on the actual PCI. Due to intermittent availability of infrastructure devices, experiments commenced when a sufficient number of devices (five out of eight) were accessible and ceased if any became unavailable. Sixteen experiments, representing over 200 hours of usage, were conducted at various time periods and with different devices, capturing variations in the application's response-time under diverse usage conditions.

We used the PSO heuristic to determin a placement that approximated optimal response-time under initial experiment conditions. Given the limited number of devices, the application's response-time was assessed from each device, all acting as clients. Each client integrates a remotely controlled request-generating process. Clients were deployed on each device, with request coordination from a remote server ensuring that the application processed only one request at a time during response-time evaluation. The global response-time was derived by averaging the response-times measured for each client. The global response-time is equivalent to the objective function of our model. In parallel to the response-time evaluation, we continuously monitored QoS parameters, including available upstream and downstream bandwidths and latency.

C. Results

Each experiment resulted in measurements of QoS parameters and response-times. We sampled them over a 2-hour period and normalized their deviation using the minimal value measured during a whole experiment. Results, depicted in Figure 3, indicated that the majority of experiments occurred in favorable network conditions, with high bandwidth availability ($> 84\%$). The stability of response-times under realistic conditions was affirmed, with 75% of samples exhibiting response-time deviations below 30%. As expected, degradation in network QoS conditions, particularly decreased bandwidth availability, led to increased response-time deviations (samples

marked as +). Some variability in results was attributed to imprecise measurements of network QoS conditions.

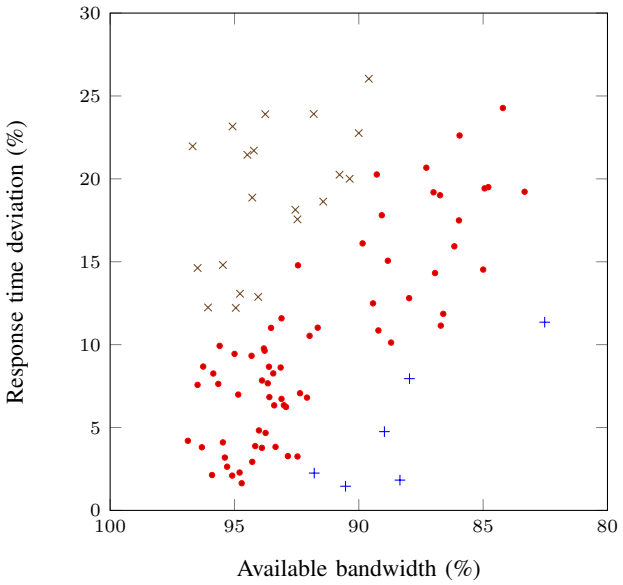

Fig. 3: Response time deviations of the application observed on different usage conditions on a real PCI.

We further investigated samples where the response-time was unexpectedly high while network conditions were favorable (marked with an × in Figure 3). These investigations revealed that some of these samples were associated with experiments where specific devices, responsible for critical services like thumbnail generation, experienced degraded network QoS conditions. Despite a majority of experiments occurring under favorable network conditions, instances of significant response-time deviations highlighted the impact of specific device conditions on overall application performance. These findings underscored the importance of considering both network conditions and the role of individual devices in optimizing the placement of microservices within a PCI.

IV. QoS-aware placement adaptation

This section focuses on exploring how dynamic adjustments to the placement of microservices can be employed to maintain acceptable application performance. We detailled our proposed solution in [13].

A. Monitoring solution

To make such adaptation relevant, specific conditions necessitating placement modification must be identified and monitored using indicators. The dynamic adaptation process involves evaluating the current placement's performance under the altered conditions. In addition to the software required for microservices deployment and operation, we integrated in the devices a monitoring tool measuring the response-times for all microservices. With such tool, we can observe time spent on each interaction between microservices in the call-graph of the application, and detect potential deviations.

The tool consists in a transparent proxy that measures the time between each request and its reply. It forwards the result to a monitoring server that centralizes all measurements so they can be used later on by the adaptation process. We need to perform at this point a calculation based on all the response-time collected for the same user request, in order to extract the specific response-times ($R_s(n,m)$) of each microservice. We tag, at the front-end, each user request with a *corellation ID* parameter, that is forwarded on all subsequent requests between microservices. Based on this value, the monitoring server is able to correlate the different response-times and calculate the different values of $R_s(n,m)$.

B. Adaptation process

Emphasizing relevance and user experience, the adaptation process should act judiciously, supervising system performance evolution and swiftly implementing placement adjustments to avoid noticeable degradation. The desired process should autonomously make decisions or rectify errors, incorporating an automatic evaluation of the chosen adaptation solution. Integrating seamlessly with the microservices orchestration system on a real PCI is crucial for practical testing and evaluation under real usage conditions.

The decision-making chain, modeled on the well-known MAPE-K architecture [6], involves monitoring, analyzing, planning, and execution. The iterative process collects real-time measurements of monitored system parameters. The analysis function derives from these values decision-making indicators. The decision phase evaluates if adaptation is necessary by applying rules defined in an adaptation strategy. Once the decision to adapt is made, the planning function determines a new placement, leveraging the PSO heuristic. The deployment function executes the chosen placement, allowing the adaptation process to reevaluate application response-time in the updated configuration, forming a continuous and dynamic adaptation loop.

C. Towards an effective adaptation strategy

1) Proactive Strategy: We tested on a first hand series of experiments a simple proactive adaptation strategy which involves regular deployments with a predefined time interval of 100 minutes. Each experiment spans for 30 hours and resulted in an average of 11 different chosen placements. We monitored the evolution of application response-times to assess the impact of different placements on performance. Notably, some adaptations result in significant improvements, while others lead to performance degradation, highlighting the dynamic nature of the system.

We explain these unsatisfactory results by considering a possible inconsistency between $R_s(n,m)$ values used for placement decision and the actual response-times measured with the new placement. Indeed, even if the $R_s(i,j)$ values 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.

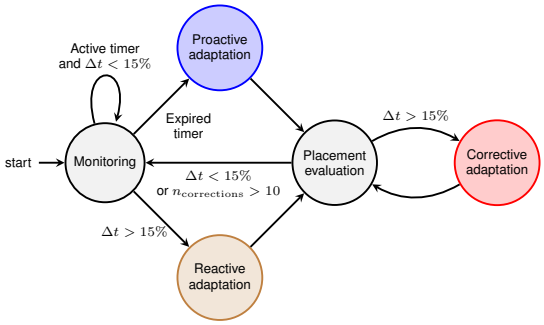

Fig. 4: Transitions in the revised adaptation strategy

2) Reactive strategy: To address this issue, we propose to integrate in our strategy the ability to compare the current response-time with previously monitored values. If a significant performance degradation is detected at any time, the process will trigger a new adaptation. This reactive behavior could occur either when monitoring the current placement between two proactive adaptation, either right after the deployment of a new placement. The first case is indicative of a change in the network QoS, the second case of an poorly chosen placement.

The revised adaptation strategy encompasses multiple stages, and the transitions between them are represented in Figure 4. The strategy evaluates transitions each time the adaptation process invokes the decision function. The proactive adaptation, triggered at regular intervals, is followed by an evaluation stage, comparing the new response-time with the pre-adaptation value. If the difference falls below the specified threshold of 15%, the placement is validated; otherwise, a corrective adaptation is initiated. The newly chosen placement is then compared to the same pre-adaptation response-time. For the sake of process completion, this evaluation is bypassed after 10 consecutive corrective adaptations.

We performed several experiments to get a representative view of the behavior of both strategies with different network QoS variations. Results show that the refined strategy presents a higher ratio of adaptations resulting in an improvement of the response-time than the proactive strategy. Thanks to corrective adaptations, new placements are more frequently deployed which helps to maintain up-to-date values for $R_s(n, m)$.

V. Conclusion

This thesis [14] provided an opportunity to investigate the unconventional challenge posed by an application deployed as microservices on domestic devices within a Participatory Cloud Infrastructure. As part of our research, we introduced a response-time model for such applications, taking into account a designated placement. Using specific response-times for each deployed service as metrics facilitated the mitigation of the model's complexity due to the interdependent nature of microservices. Through simulation, we validated a heuristic for placement selection, that gives a response-time approximation near the optimal solution, with a maximum deviation of 5%, with a reduced execution time.

Tests of our solution on a real Participatory Cloud Infrastructure highlighted the need to cope with the dynamic nature of network QoS. A degradation of these parameters on a single device could have a comprehensive impact on the application's response-time. As a solution, we presented an adaptation framework able to trigger new deployments of the application when such degradation is detected. Our solution gave us satisfactory results by adding the ability to dynamically evaluate and, if necessary, correct the proposed placement.

This thesis suggests possible future works related to automatically adjusting the number of microservices' instances to manage increased traffic. Additionally, we identified various security issues in our scenario linked to a participatory system.

References

[1] Ozalp Babaoglu, Moreno Marzolla, and Michele Tamburini. Design and Implementation of a P2P Cloud System. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12.

[2] Vincenzo D. Cunsolo, Salvatore Distefano, Antonio Puliafito, and Marco Scarpa. Cloud@Home: Bridging the Gap between Volunteer and Cloud Computing. In *Emerging Intelligent Computing Technology and Applications*, Lecture Notes in Computer Science, 2009.

[3] Daniel J. Dubois, Giuseppe Valetto, Donato Lucia, and Elisabetta Di Nitto. Mycocloud: Elasticity through Self-Organized Service Placement in Decentralized Clouds. In *2015 IEEE 8th International Conference on Cloud Computing*, June 2015.

[4] 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.*

[5] C. Jatoth, G. R. Gangadharan, and R. Buyya. Computational Intelligence based QoS-aware Web Service Composition: A Systematic Literature Review. *IEEE Transactions on Services Computing*, 2015.

[6] Jeffrey Kephart and William Walsh. An architectural blueprint for autonomic computing. Technical report, IBM, 2003.

[7] Amin M. Khan and Felix Freitag. On Edge Cloud Service Provision with Distributed Home Servers. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom).*

[8] Wenfeng Li, Ye Zhong, Xun Wang, and Yulian Cao. Resource virtualization and service selection in cloud logistics. *Journal of Network and Computer Applications*, November 2013.

[9] Tessema M. Mengistu, Abdulrahman M. Alahmadi, Yousef Alsenani, Abdullah Albuali, and Dunren Che. cuCloud: Volunteer Computing as a Service (VCaaS) System. In Min Luo and Liang-Jie Zhang, editors, *Cloud Computing – CLOUD 2018*, Lecture Notes in Computer Science.

[10] M. Ryden, K. Oh, A. Chandra, and J. Weissman. Nebula: Distributed Edge Cloud for Data Intensive Computing. In *2014 IEEE International Conference on Cloud Engineering*.

[11] Mennan Selimi, Llorenç Cerdà-Alabern, Marc Sánchez-Artigas, Felix Freitag, and Luís Veiga. Practical Service Placement Approach for Microservices Architecture. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID).*

[12] Bruno Stévant, Jean-Louis Pazat, and Alberto Blanc. Optimizing the Performance of a Microservice-Based Application Deployed on User-Provided Devices. In *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC).*

[13] Bruno Stévant, Jean-Louis Pazat, and Alberto Blanc. QoS-aware Autonomic Adaptation of Microservices Placement on Edge Devices. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science*, 2020.

[14] Bruno Stévant. Vers une infrastructure participative d'hébergement de services à destination des communautés virtuelles : Orchestration dynamique de micro-services selon les conditions d'utilisation. PhD Thesis, INSA de Rennes, 2022. https://theses.hal.science/tel-04522400