IRENE: Interference and High Availability Aware Microservice-based Applications Placement for Edge Computing

Paulo Souza[1][a], João Nascimento[1][b], Conrado Boeira[1][c], Ângelo Vieira[1][d], Felipe Rubin[1][e], Rômulo Reis[1][f], Fábio Rossi[2][g] and Tiago Ferreto[1][h]

[1] *Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil*

[2] *Federal Institute of Education, Science and Technology Farroupilha, Alegrete, Brazil*

Keywords: Edge Computing, Microservices, Genetic Algorithm, Applications Placement.

Abstract: The adoption of microservice-based applications in Edge Computing is increasing, as a result of the improved maintainability and scalability delivered by these highly-distributed and decoupled applications. On the other hand, Edge Computing operators must be aware of availability and resource contention issues when placing microservices on the edge infrastructure to avoid applications' performance degradation. In this paper, we present IRENE, a genetic algorithm designed to improve the performance of microservice-based applications in Edge Computing by preventing performance degradation caused by resource contention and increasing the application's availability as a means for avoiding SLA violations. Experiments were carried, and the results showed IRENE effectiveness over different existing strategies in different scenarios. In future investigations, we intend to extend IRENE's interference awareness to the network level.

1 INTRODUCTION

Edge Computing is gaining attention by delivering processing power to mobile applications with reduced latency compared to cloud solutions (Satyanarayanan, 2017). For this goal, small-sized data centers, also referred to as cloudlets, are positioned at strategic positions to provide fast feedback while avoiding core network saturation. While the proximity between cloudlets and end-users allows for reduced response times, edge infrastructure operators still have resource management concerns regarding the placement of applications. These concerns are even more challenging for applications built upon the microservices architecture, which are designed with a particular set of technologies to get the best-of-breed out of modern computing infrastructure.

Microservice-Based applications usually prioritize smaller footprint, reduced storage requirements, and the faster boot time of lightweight virtualization over reinforced isolation provided by traditional virtual machines (Gannon et al., 2017). As a consequence, the co-location of multiple microservices into the same server may result in performance-degrading events due to resource contention.

Unlike most of cloud data centers, cloudlets may count on heterogeneous servers to host applications, which may provide different performance and availability assurances. Therefore, choosing host servers for accomodating microservices also involves considering applications' availability requirements as a means to avoid Service Level Agreements (SLAs) violations.

Several studies have been focusing on optimizing the placement of applications (Wang et al., 2012; Romero and Delimitrou, 2018). Whereas some of them provided solutions for ensuring high availability, others have focused on avoiding performance interference. However, providing both minimal performance interference and high availability for applications' placement remains an open problem.

Therefore, in this paper, we present IRENE, a genetic algorithm designed for ensuring *minimal inter-*

[a] https://orcid.org/0000-0003-4945-3329
[b] https://orcid.org/0000-0002-2261-2791
[c] https://orcid.org/0000-0002-6519-9001
[d] https://orcid.org/0000-0002-1806-7241
[e] https://orcid.org/0000-0003-1612-078X
[f] https://orcid.org/0000-0001-9949-3797
[g] https://orcid.org/0000-0002-2450-1024
[h] https://orcid.org/0000-0001-8485-529X

Souza, P., Nascimento, J., Boeira, C., Vieira, Â., Rubin, F., Reis, R., Rossi, F. and Ferreto, T.
IRENE: Interference and High Availability Aware Microservice-based Applications Placement for Edge Computing.
DOI: 10.5220/0009581604900497
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 490-497
ISBN: 978-989-758-424-4

ference to avoid performance degradation and *high availability* as a means for minimizing SLA violations during the placement of microservice-based applications. We performed a set of experiments that showed that IRENE could overcome several existing strategies.

The remaining of this paper is organized as follows. In Section 2, we discuss the emergence of Edge Computing, the popularization of microservice-based applications, and challenges faced by edge infrastructure operators during the placement of these applications, especially regarding performance interference and high availability. We also complement the background section with an overview of current investigations in Section 3, which also discusses the research gap tackled by our study. After describing the problem we aim to solve in Section 4, we introduce IRENE, our proposed genetic algorithm. Finally, Sections 6 and 7 are reserved for the evaluation of IRENE versus three baseline heuristic placement strategies, as well as for final remarks and directions for future investigations.

2 BACKGROUND

The increasing amount of sensors integrated on multiple devices allowed the rise of various Internet of Things (IoT) applications, such as Smart Homes and Smart Cities (Shi et al., 2016). Due to the intrinsic limitation of embedded devices used in IoT applications in terms of resources, it becomes infeasible to perform some computational tasks locally (Satyanarayanan et al., 2009). A solution would be to send the data collected to the cloud. Nonetheless, transporting data to the cloud introduces new challenges, such as network latency and security. There are also connectivity restrictions that make the task of sending the data from IoT and mobile applications to the cloud even more challenging.

The concept of Edge Computing was proposed as a solution to this problem. Edge Computing performs computations at the edge of the network, closer to the devices that are producing/consuming the data (Shi et al., 2016). It results in a reduction of network latency, as the data does not need to travel long distances to reach the cloud. Satyanarayanan et al. (Satyanarayanan et al., 2009) proposed the concept of cloudlets as simpler and smaller data centers that can be responsible for the computation in the Edge Computing.

Microservices architecture is a recommendation for getting the maximum benefits from the Cloud Computing (Gannon et al., 2017), and this recommendation also fits well for Edge Computing. The microservices architecture proposes an opposing view to the monolithic architecture by segmenting the different application functionalities into multiple components. Each microservice can be managed independently from others and has limited responsibilities and dependencies (Gannon et al., 2017). They can also be accommodated in cloudlets within the Edge Computing paradigm as a means for allowing improved use of resources. Two of the main factors that edge infrastructure operators must keep in mind during the decision of where to deploy the microservices are high availability and interference.

When deploying an application on cloudlets, we want the system to be highly available; that means, to keep working properly, even when one or more servers are not responding. Depending on how available an application is, the operator can provide Service Level Agreements (SLAs) to the users. These are explicit or implicit contracts that define service levels promised in the form of a metric of interest like availability, for example. SLAs also provide specific remedies in case of any unexpected event.

Another factor that needs to be considered when deciding the applications' placement is how different microservices in the same host interfere with each other. At this point, understanding the core technologies such as virtualization, which responsible for hosting applications inside servers, is necessary. Virtualization allows microservices to obtain access to shared resources from the same host while still keeping a certain degree of isolation during their execution. There are two types of virtualization: hypervisor-based and container-based.

In the hypervisor-based virtualization, each application has its own virtual machine with a guest operating system. The container-based virtualization leverages OS-level features to provide isolation among applications that share the host kernel as a means for improved performance (Soltesz et al., 2007). The container-based virtualization has the advantage of being lightweight (Gannon et al., 2017), which makes it more suitable for Edge Computing, where hosts present resource constraints. However, containers have the downside that they grant less isolation for the applications, compared to hypervisor-based virtualization.

Applications tend to require more intensively a specific computational resource (e.g., CPU, memory, etc.) (Qiao et al., 2017). Microservices on the same host using the same resources may end up degrading each other performance. In an ideal scenario, where applications could run on dedicated servers, there would be no interference. However, this scenario is usually not real due to the virtualization, which allows a server to share its computational resources among several microservices, which must compete for the shared resources. This competition can be avoided during the choice of where to accommodate applications. A remedy to avoid performance interference would be considering the placement of microservices with non-conflicting resource bounds (Qiao et al., 2017).

3 RELATED WORK

3.1 High Availability

Bin et al. (Bin et al., 2011) propose a heuristic algorithm that monitors the infrastructure to detect failures beforehand and tries to reallocate applications from hosts before they fail, thus enhancing applications' availability.

Zhu and Huang (Zhu and Huang, 2017) formulate a stochastic model for adaptive placement of Edge applications to achieve lower costs, while still considering high availability. As the problem scales, to circumvent the complexity of computations, a heuristic algorithm is also presented for real-world scenarios.

Wang et al. (Wang et al., 2012) propose a high availability scheme for virtual machines in cloud data centers. The authors take advantage of both vertical and horizontal scalability to improve applications' availability. To this end, the number of instances per application is increased, enabling their placement in other hosts, thus improving availability.

3.2 Performance Interference

Kim et al. (Kim et al., 2018) propose a management mechanism to reduce the memory interference latency. The proposed approach differentiates the execution of tasks by separating them into critical and normal control groups. When a critical task is running, a prediction about excessive memory consumption is made. If deemed necessary, the CPUFreq governor is activated to throttle memory requests made by applications of the normal control group.

Ren et al. (Ren et al., 2019) present a machine learning-based prediction framework that enables the classification of each task as repetitive or new. In contrast to new tasks, repetitive ones can provide historical information regarding their behavior, which can be used to improve their scheduling to achieve performance gains.

Romero and Delimitrou (Romero and Delimitrou, 2018) present a way to optimize performance and efficiency in systems with intra-server and inter-server heterogeneity, with a solution called Mage. Mage continuously monitors the performance of active applications and minimizes resource contention in cloud systems. Using data mining, it explores the space of application placements and determines those that minimize interference between co-located applications.

3.3 Our Contributions

Several studies have been focused on optimizing availability or performance interference requirements. However, none of them considered both objectives during the placement of applications in edge scenarios. Therefore, in this paper, we propose IRENE, a genetic algorithm for improving the placement of microservice-based applications regarding both high availability requirements as means for minimizing the number of SLA violations and interference that leads to superior applications' performance.

4 PROBLEM FORMULATION

In this study, we consider an Edge Computing scenario where applications containing one or more microservices need to be placed inside physical servers on a cloudlet. Our goal is to design a satisfactory placement for applications considering a twofold objective:

1. Minimizing performance interference by avoiding the co-location of microservices with similar resource bounds (e.g., CPU-bound) in the same host.
2. Minimizing the number of SLA violations by distributing applications' microservices on hosts that satisfy its availability requirements.

Table 1 presents the list of terminology adopted in this study. We demonstrate both the capacity of physical servers and the demand of microservices as a resource vector containing: (i) CPU (number of cores); (ii) memory (Gigabytes); and (iii) storage (Gigabytes). The set of g physical servers is denoted by $\mathbb{P} = \{P_1, P_2, ..., P_g\}$. Each physical server P_i provides an availability denoted by $\alpha(P_i)$ and has a binary variable $u(P_i)$, which represents whether the server is being used or not (0 = inactive, 1 = active).

We define the set of h applications as $\mathbb{A} = \{A_1, A_2, ..., A_h\}$, where each microservice $A_j \in \mathbb{A}$ has a placement requirement $\beta(A_j)$ that represents the minimum availability ratio expected by A_j. In this scenario, each application $A_j \in \mathbb{A}$ is composed of

Table 1: List of terminology used in this paper.

Notation	Description
$\mathbb{P}$	Set of g servers within the cloudlet
$\mathbb{A}$	Set of h applications
$\mathbb{M}$	Set of p microservices to be hosted
$\alpha(P_i)$	Variable that represents the availability level assured by a server P_i
$\beta(A_j)$	Variable that represents the minimum availability ratio expected by an application A_j
$\sigma(P_i)$	Variable that represents the number of colliding microservices inside a server P_i
$u(P_i)$	Binary variable that receives 1 if a server P_i is active or or receives 0 otherwise
$x_{i,k}$	Binary variable that receives 1 if a server P_i hosts a microservice k, or receives 0 otherwise

one or more microservices. The set of all p microservices within the cloudlet is defined as $\mathbb{M} = \{M_1, M_2, ..., M_p\}$.

Each microservice may be distributed across the hardware components differently. For simplicity reasons, we consider microservices as CPU-bound, memory-bound, or IO-bound. We also define which microservices belong to which applications with the $v_{k,j}$ variable, that receives 1 if a microservice M_k is part of an application A_j, and receives 0 otherwise.

5 PROPOSAL

The problem we aim to solve in this work can be considered a variant of the vector bin-packing problem. It can be briefly formulated as packing n vectors into k sets (or bins) without surpassing their limit. This problem is acknowledged to be an NP-hard problem (Zhang et al., 2010), and hence, no optimal solution in polynomial time is known. Therefore, this work proposes a solution that uses a Genetic Algorithm, as it offers a near-optimal solution in constrained time.

Genetic Algorithms (GA) are a class of algorithms that are based on Darwin's ideas of natural selection and evolution. It encodes possible solutions for a problem into the chromosome-like structure and then applies recombination methods. The idea is that a GA starts with a population of potential solutions characterized as individuals. Then, it evaluates each one of them accordingly to a defined set of mathematically expressed goals, which ones are better and worse solutions. The collection of mathematical equations used to evaluate solutions is called fitness function. The individuals deemed "fitter" have attributed to them a higher chance of "reproducing", as in passing some of its characteristics to a new individual (Whitley, 1994). Therefore, GA can reach a satisfactory solution by repeating this process multiple times.

5.1 Chromosome Representation

In the genetic algorithm presented in this work, the genes of an individual represent which server will host each microservice. In other words, if bit i has value x, it means the microservice with id i will be hosted in server number x. Therefore, each individual represents a possible placement configuration.

5.2 Fitness Function

IRENE assesses solutions based on the trade-off between cloudlet's consolidation rate, applications' availability, and microservices interference. To achieve this goal, IRENE utilizes the following fitness function to evaluate each solution z on the population:

$$\mathfrak{f}(z) \leftarrow (\mathfrak{c} + \mathfrak{s} + \mathfrak{h})^2 \tag{1}$$

The $\mathfrak{c}$ variable represents the colliding microservices ratio resulted from the solution. To calculate this ratio, IRENE applies a "rule of three" equation to measure the representativeness of the number of colliding microservices gathered by the σ function concerning the number of p microservices within the cloudlet. This process is depicted next:

$$\mathfrak{c} \leftarrow 100 - \left(\frac{(\sum_i^{\mathbb{P}} \sigma(p_i)) \times 100}{p}\right) \tag{2}$$

The $\mathfrak{s}$ variable gets the number of applications whose SLA requirements were violated with the proposed placement. This process involves checking the ϑ function, that verifies if the SLA of an application was violated. To accomplish this, the ϑ function compares the guaranteed availability ratio provided by each server through the α function against the application's availability requirements:

$$\mathfrak{s} \leftarrow 100 - \left(\frac{(\sum_i^{\mathbb{A}} \vartheta(a_i)) \times 100}{h}\right) \tag{3}$$

The $\mathfrak{h}$ represents the cloudlet's consolidation rate. This variable utilizes a "rule of three" equation to calculate how many servers stay inactive within the cloudlet after the suggested placement is applied. This process is described in the following equation:

$$\mathfrak{h} \leftarrow 100 - \left(\frac{(\sum_i^{\mathbb{P}} u(p_i)) \times 100}{g}\right) \tag{4}$$

To evaluate the generated solutions, IRENE sums $\mathfrak{c}$, $\mathfrak{s}$, and $\mathfrak{h}$ variables, and the result of this sum is raised to the power of 2. We perform this exponentiation to aid the genetic algorithm to understand that even a slightly better solution may refer to significant progress in overall.

5.3 Genetic Operators

In this section, we present IRENE's genetic operators. We used the Roulette Wheel Selection method as the selection algorithm for our approach. In this procedure, the probability of an individual being chosen as a parent is directly proportional to its fitness score. The highest the fitness rate, the highest the likelihood of a chromosome being selected for the list of parents used in the selection phase (Sivaraj and Ravichandran, 2011).

The next step, after the parents' selection, the genetic algorithm framework proceeds to the crossover operation. This stage consists of generating the offspring by combining parts of the genes from the selected parents. There are multiple methods to perform the crossover operation, but the one chosen was a uniform crossover. The uniform crossover consists of randomly choosing from which parent the genes are copied from (Umbarkar and Sheth, 2015). During this step, a random mutation is also performed. Any offspring has a specific probability of being mutated and having one of its bits randomly chosen to be mutated. This means that when a solution is mutated, one of the microservices is assigned to a random cloudlet server.

6 NUMERICAL EVALUATION

6.1 Workloads Description

For the evaluation, we consider a data set with a fixed number of 18 applications, grouped according to their SLA requirements and number of microservices (also called as application size for brevity purposes).

Regarding the SLA requirements, we divide applications in three groups according to the minimum acceptable availability rate: *(a) Low SLA Requirement:* 80%; *(b) Medium SLA Requirement:* 85%; and *(c) High SLA Requirement:* 90%. Regarding the number of microservices, we also divide applications in three groups: (i) *Small:* 3 microservices; (ii) *Medium:* 6 microservices; and (iii) *Large:* 9 microservices.

Given the heterogeneous nature of edge environments, where a multitude of applications with different behaviors must be allocated, in this study, we concentrate on assessing the implications of having to accommodate applications with different sizes (Table 4 and SLA requirements (Table 5). To aid the analysis, we also consider a baseline scenario (Table 6) wherein applications are divided equally regarding number of microservices and SLA requirements. During the evaluation, we considered microservices that may have different behaviors, based on the resource bounds (e.g., CPU-Bound, Memory-Bound, and IO-Bound). The resource demands of microservices may also be different, as can be seen in Table 2.

Table 2: Different microservice resource demands considered during the evaluation.

Size	CPU (# of Cores)	Memory (GB)	Disk (GB)
Small	1	1	8
Medium	2	2	16
Large	4	4	32

We consider a fixed number of 60 homogeneous edge servers with the following configuration: *CPU*: 16 cores, *RAM:* 16GB, *Disk:* 128GB. The servers are divided according to different levels of availability they can assure, as described in Table 3.

Table 3: Availability levels of edge servers.

Type	Availability	Number of servers
Low	90%	30
Medium	95%	20
High	99.9%	10

We compare our proposal against the following algorithms:

- **Best-Fit (BF):** Prioritizes server consolidation, reducing the total number of servers used to host applications.
- **Worst-Fit (WF):** Focuses on balancing workloads between servers.
- **IntHA:** focuses on accommodating applications in a minimal number of servers given interference and high availability goals. To accomplish this, IntHA takes into account both the assured availability (depicted by $\alpha(P_i)$) and the number of colliding microservices (represented by $\sigma(P_i)$) for each host. Algorithm 1 shows the pseudocode for this heuristic.

All solutions considered during the evaluation were developed and executed using the Ruby language v2.7.0p0 (2019-12-25 revision 647ee6f091). For the execution of the solutions, we used a Linux host machine with specifications contained in Table 7. During the experiments, we configured IRENE to look for placement solutions during 25000 generations using a population of 500 chromosomes, with mutation rate set to 35% and 100 parents being used for mating. All experimentation assets can be found at our GitHub repository (https://github.com/paulosevero/IRENE).

Table 4: Application size scenarios.

		Small-Size Applications	*Medium-Size Applications*	*Large-Size Applications*
Scenario 1	*Low SLA Requirement*	1	1	4
	Medium SLA Requirement	1	1	4
	High SLA Requirement	1	1	4
	TOTAL	3	3	12
Scenario 2	*Low SLA Requirement*	4	1	1
	Medium SLA Requirement	4	1	1
	High SLA Requirement	4	1	1
	TOTAL	12	3	3

Table 5: SLA requirements scenarios.

		Small-Size Applications	*Medium-Size Applications*	*Large-Size Applications*
Scenario 1	*Low SLA Requirement*	1	1	1
	Medium SLA Requirement	1	1	1
	High SLA Requirement	4	4	4
	TOTAL	6	6	6
Scenario 2	*Low SLA Requirement*	4	4	4
	Medium SLA Requirement	1	1	1
	High SLA Requirement	1	1	1
	TOTAL	6	6	6

Table 6: Baseline scenario.

		Small-Size Applications	*Medium-Size Applications*	*Large-Size Applications*
Baseline	*Low SLA Requirement*	2	2	2
	Medium SLA Requirement	2	2	2
	High SLA Requirement	2	2	2
	TOTAL	6	6	6

Algorithm 1: IntHA heuristic.

```
$\mathbb{A} \leftarrow \{A_1, A_2, ..., A_h\}$
$\mathbb{P} \leftarrow \{P_1, P_2, ..., P_g\}$
$\mathbb{M} \leftarrow \{M_1, M_2, ..., M_p\}$
foreach application $A \in \mathbb{A}$ do
    foreach microservice $M \in \mathbb{M}$ do
        if $M \in A_{ms}$ then
            $\mu \leftarrow null$
            foreach server $P \in \mathbb{P}$ do
                if $P$ has cap. to host $M$ then
                    $s \leftarrow \frac{P_{usage} + \alpha(P)}{1 + \sigma(P)}$
                end
                if $s \geq \mu_{score}$ then
                    $\mu \leftarrow P$
                end
            end
            $\mu_{ms} \leftarrow \mu_{ms} \cup M$
        end
    end
end
```

6.2 Results and Analysis

6.2.1 Application Size

The goal of this analysis is to understand how much the average size of applications impact the evaluated solutions behavior regarding interference, SLA violations, and consolidation rate. The results of this analysis are depicted in Figure 1.

Table 7: Testbed specifications.

Component	Specification
Processor	Intel(R) Core(TM) i7-4790 4(8) @ 3.60GHz
Memory	16GB DDR3 @ 1333MHz
Storage	1TB HDD SATA @ 7200RPM
Operation System	elementary OS 5.1.2 Hera

Achieving high levels of availability becomes harder when applications have more microservices. Therefore, facing a scenario with larger applications leads to more SLA violations. Consolidating applications inside hosts that provide higher availability guarantees avoid SLA violations, but with the cost of causing the collision of more microservices. This explains why the Best-Fit approach surpasses Worst-Fit and IntHA when handling SLA violations, but loses in terms of interference. IRENE, on the other hand, follows a different path for avoiding SLA violations without hurting too much the other metrics. It focuses on not wasting the resources of servers that provide higher availability with applications that have low SLA requirements.

In the evaluated scenarios, we can notice that there is a trade-off between delivering high levels of availability and avoiding collision of microservices with similar resource bounds. If we focus on providing the highest availability possible for all applications, we will be forced to consolidate all the workload inside a limited number of servers that offer higher guarantees. However, this decision would lead to several microservices with similar bound being hosted by the same server, which raises concerns with interference.

As a consequence, considering just one of these

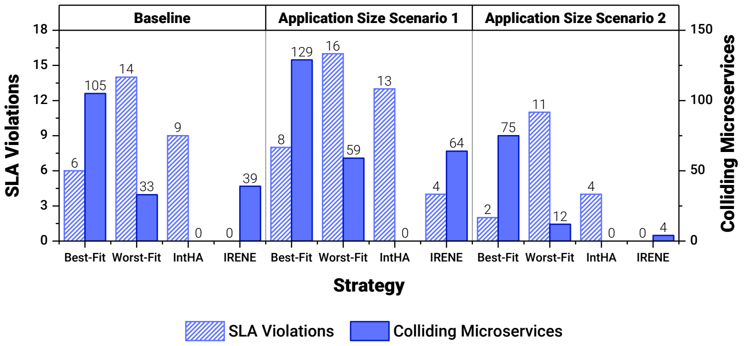

Figure 1: Heuristic results in edge scenarios with variable application sizes.

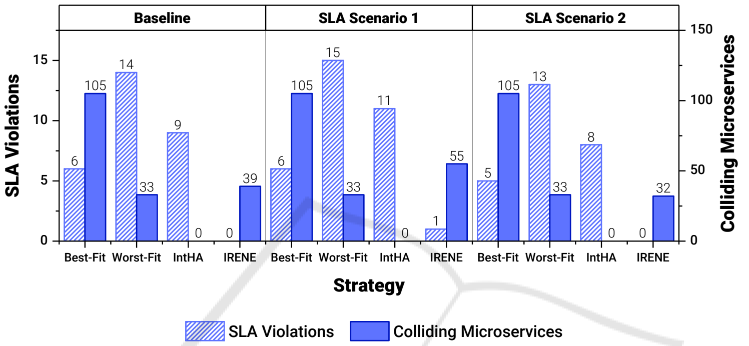

Figure 2: Heuristic results in edge scenarios with variable application SLA requirements.

two metrics may result in placements that hurt the other. For example, Best-Fit and Worst-Fit only take into account the number of hosts used for accommodating applications. Best-Fit tries to consolidate applications the most and ends up by causing high interference levels. On the other hand, Worst-Fit that tries to balance the workload the most ends up by providing poor availability ratios which result in several SLA violations.

Bearing in mind this trade-off, IRENE chooses to sacrifice the consolidation rate a little to achieve more significant gains regarding interference and SLA violations. Compared to Best-Fit, it sacrifices the consolidation rate in 16% for the sake of reducing the number of colliding microservices in 45.74% and minimizing the SLA violations by half.

6.2.2 SLA Requirements

This analysis aims at examining the implications of decisions made by placement strategies regarding interference, consolidation rate, and SLA violations in edge scenarios containing applications with variable SLA requirements. The results of this analysis is depicted in Figure 2.

When most of applications have high availability requirements, placement strategies may have to stack as many microservices as possible into servers that provide higher availability in order to avoid SLA violations. As a consequence, Best-Fit suits well in this scenario, being able to achieve the second-best result regarding SLA violation while utilizing only 16 servers. However, due to the order of placement of applications being crescent in availability need, the Best-Fit heuristic guarantees a higher SLA for the low availability applications and a lower one for the applications that demand a higher availability.

In contrast, IRENE is able to minimize the percentage of applications with SLA violations in 27.77% compared to Best-Fit in the first scenario. Furthermore, IRENE manages to have 0 SLA violations in scenario 2. However, this gain is achieved at the cost of reducing the consolidation rate in 16.66% and 20.0% for scenarios 1 and 2 respectively.

Another aspect where IRENE differ from Best-Fit is the SLA guarantees for each type of applications. Our algorithm ensures higher SLAs for applications that need higher availability and lower SLAs for the ones with lower availability needs. This decision of stacking as many microservices from applications with high SLA requirements as possible into the subset of servers that provide higher availability guarantees also allows IRENE to accommodate, in scenario 1, all microservices in only 26 servers, 2 less than in the baseline scenario. Nonetheless, it also generates an increase of 14.81% on the number of co-located microservices with similar bound compared to the result it achieved in the baseline scenario.

7 CONCLUSIONS

Edge Computing is gaining significant popularity with the idea of using small-sized data centers (often called as cloudlets) to bring data processing closer to end-users. Even though cloudlets provide faster response time, performance-degrading events such as resource contention may affect applications' performance and, consequently, negatively influence the end-user experience. This performance interference can be even more dangerous considering an existing trend in modern application development of prioritizing flexibility provided by microservices running on containers over the improved isolation offered by the classical approach that uses virtual machines. On scenarios such as those, high availability requirements present in SLAs also come into the scene. When placing all microservices of a given application on a single host, it becomes a single point of failure.

Previous investigations proposed solutions for high availability issues or performance interference demands over cloud-based applications. However, none of them focused on providing a solution for both objectives in edge scenarios. Therefore, in this paper, we present IRENE, a genetic algorithm approach designed to acquire the best of breed out of edge servers regarding high availability (as means for avoiding SLA violation) and performance interference (to achieve superior application performance) during the placement of microservice-based applications. We validated IRENE through a set of experiments, and the results showed that it could overcome several existing approaches with minimal overhead. As future work, we intend to minimize performance interference issues at the network level by reducing packets collision and network saturation.

ACKNOWLEDGEMENTS

This work was supported by the PDTI Program, funded by Dell Computadores do Brasil Ltda (Law 8.248 / 91).

REFERENCES

Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E. K., Moatti, Y., and Lorenz, D. H. (2011). Guaranteeing high availability goals for virtual machine placement. In *2011 31st International Conference on Distributed Computing Systems*, pages 700–709. IEEE.

Gannon, D., Barga, R., and Sundaresan, N. (2017). Cloud-native applications. *IEEE Cloud Computing*, 4(5):16–21.

Kim, J., Shin, P., Noh, S., Ham, D., and Hong, S. (2018). Reducing memory interference latency of safety-critical applications via memory request throttling and linux cgroup. In *2018 31st IEEE International System-on-Chip Conference (SOCC)*, pages 215–220. IEEE.

Qiao, S., Zhang, B., and Liu, W. (2017). Application classification based on preference for resource requirements in virtualization environment. In *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 176–182. IEEE.

Ren, S., He, L., Li, J., Chen, Z., Jiang, P., and Li, C.-T. (2019). Contention-aware prediction for performance impact of task co-running in multicore computers. *Wireless Networks*, pages 1–8.

Romero, F. and Delimitrou, C. (2018). Mage: Online interference-aware scheduling in multi-scale heterogeneous systems. *arXiv preprint arXiv:1804.06462*.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.

Satyanarayanan, M., Bahl, V., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

Sivaraj, R. and Ravichandran, T. (2011). A review of selection methods in genetic algorithm. *International journal of engineering science and technology*, 3(5):3792–3797.

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., and Peterson, L. (2007). Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 275–287. ACM.

Umbarkar, A. and Sheth, P. (2015). Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1).

Wang, W., Chen, H., and Chen, X. (2012). An availability-aware virtual machine placement approach for dynamic scaling of cloud applications. In *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pages 509–516. IEEE.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18.

Zhu, H. and Huang, C. (2017). Availability-aware mobile edge application placement in 5g networks. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE.