Latency-Aware Cost-Efficient Provisioning of Composite Applications in Multi-Provider Clouds

Daniel Chaves Temp[1,3][a], Igor Ferrazza Capeletti[3][b], Ariel Goes de Castro[3][c], Paulo Silas Severo de Souza[2][d], Arthur Francisco Lorenzon[4][e], Marcelo Caggiani Luizelli[3][f] and Fábio Diniz Rossi[1,3][g]

[1] *Federal Institute Farroupilha, Alegrete, Brazil*
[2] *Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil*
[3] *Federal University of Pampa, Alegrete, Brazil*
[4] *Federal University of Rio Grande do Sul, Porto Alegre, Brazil*

Keywords: Cloud, Cost, Heuristics, Latency, Service Placement, Simulation.

Abstract: Composite applications have the versatility of maintaining several functions and services geographically distributed but being part of the same application. This particular software architecture fits in very easily with the model of distributed regions present in most cloud players. In this way, the search for leasing applications at the lowest cost becomes a reality, given that the application services can be in different players at a lower cost, as long as the performance metrics of the application as a whole are met. Performing provisioning decisions considering the allocation cost of different providers and the latency requirements of applications is not trivial, as these requirements are often conflicting, and finding good trade-offs involves the analysis of a large-scale combinatorial problem. Accordingly, this paper presents *Clover*, a placement algorithm that employs score-based heuristic procedures to find the best provisioning plan for hosting composite applications in geographically distributed cloud environments. Simulated experiments using real latency traces from Amazon Web Services indicate that *Clover* can achieve near-optimal results, reducing latency issues and allocation cost by up to 74.47% and 21.2%, respectively, compared to baseline strategies.

1 INTRODUCTION

In the context of cloud computing, composite applications (or decoupled applications) are built using, in general, microservices architecture (Katsuno and Takahashi, 2015). In a microservices architecture (Raj et al., 2023), an application is divided into smaller, independent services that can be developed, deployed, and scaled independently of each other. It allows for greater flexibility and faster development and deployment cycles, as each service can be updated or changed without affecting the other services. Figure 1 presents an evolution of software architectures, starting from the monolith in which all software functions and components are compiled into a single block, which makes maintenance and scalability issues more difficult. Software-Oriented Architectures (SOA) allow the separation of business rules and interfaces so that the application is more flexible in terms of scalability and maintenance, as the components can be treated independently. Microservices consist of an evolution of SOA, which operates at a smaller granular level, and functions can be instantiated and released on demand, allowing high performance in addition to fast scalability and independent maintenance. In SOA (Ilie et al., 2022), the decomposition of the application into individual services now allows creating, testing, and tuning of services simultaneously, eliminating monolithic development cycles. The problem is that centralizing communication on an Enterprise Service Bus (EBS) (Chaudhari et al., 2016) represented a single point of failure for the entire system—basically, another monolithic

[a] https://orcid.org/0000-0002-9724-1331
[b] https://orcid.org/0000-0002-8712-2195
[c] https://orcid.org/0000-0002-5391-5082
[d] https://orcid.org/0000-0003-4945-3329
[e] https://orcid.org/0000-0002-2412-3027
[f] https://orcid.org/0000-0003-0537-3052
[g] https://orcid.org/0000-0002-2450-1024

Temp, D., Capeletti, I., Goes de Castro, A., Severo de Souza, P., Lorenzon, A., Luizelli, M. and Rossi, F.
Latency-Aware Cost-Efficient Provisioning of Composite Applications in Multi-Provider Clouds.
DOI: 10.5220/0011995600003488
In *Proceedings of the 13th International Conference on Cloud Computing and Services Science (CLOSER 2023)*, pages 297-305
ISBN: 978-989-758-650-7; ISSN: 2184-5042

structure that could congest the whole development cycle. Unlike services in SOA, microservices can communicate with each other, usually stateless, making software more fault tolerant and less dependent on a single ESB. In addition, microservices can communicate through application programming interfaces (APIs). In terms of cost and scalability, the advantage of this type of application lies on these microservices may be hosted on different servers, in different locations, and may even be built using different technologies, designed to communicate with each other and work together to provide the desired functionality (Blinowski et al., 2022). As this type of application can be easily scaled and modified as needed without requiring a complete overhaul of the entire application, it makes them well-suited for use in cloud computing environments, where resources can be added or removed as needed to meet changing demand.

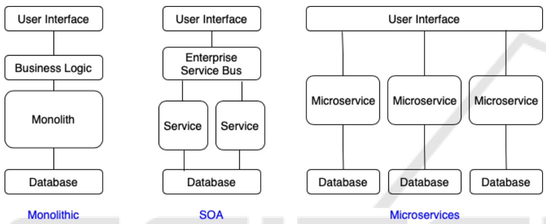


Figure 1: Software architectures.

Even more so, it is the ideal scenario for composite applications when we are working on cloud players that maintain data centers in geographically distributed regions but still maintain acceptable levels of performance in terms of network latency between regions, because a composite application presents a feature that can be used to reduce costs (Vu et al., 2019). As the software architecture is decoupled, each piece of software can be placed in a geographically different location as long as these entities' latencies do not exceed the service level agreement established to the application (Pelle et al., 2019). This paper presents *Clover*, a placement algorithm that employs score-based heuristic procedures to reduce the allocation cost incurred by provisioning composite applications in geo-distributed cloud environments without overlooking the application latency requirements. In summary, we make the following contributions: (i) We present a conceptual model that captures the cost and latency requirements considered during the provisioning of composite applications in geo-distributed clouds (§4), (ii) we propose a novel placement algorithm (*Clover*) that introduces score-based functions for reducing allocation cost and latency issues during the provisioning of composite applications (§5), and (iii) we conduct simulated experiments using real latency values from Amazon Web Services that demonstrate that *Clover* achieves near-optimal results, reducing cost and latency issues by up to 74.47% and 21.2%, respectively, compared to baseline strategies (§6).

2 BACKGROUND

Cloud environments (De Donno et al., 2019) have become a de facto standard in providing services via the Internet. Services are distributed via the Internet through a cloud services platform with per-use pricing. Paying only for what the customer uses helps to reduce operating costs, run processes more efficiently, and make changes as the business needs evolve. One of the most important benefits supplied by cloud computing is the capacity to scale elastically, regardless of geographic origin (Liu et al., 2018). It can provide a much better experience to your customers simply and quickly. But it is not always offering a minimal cost, as it could be. Depending on the choices made by the cloud operators, whoever deployed the service may be paying more than they should. Public cloud providers offer multiple regions that serve as access points distributed worldwide. These regions deliver for services to be in places closer to customers in order to decrease latency in accessing services. However, the values for maintaining a hosted service differ from region to region (Kozina and Kozin, 2022). It is due to the local cost of energy consumption, air conditioning, and legislation, among others. It would always be advantageous to host services in the cheapest regions if not for the latency cost imposed by the distance that the region may be from the customer. The inverse is also a problem because if the cloud operators only consider the service's latency, they may be paying more to maintain a minimum latency much higher than that needed to offer the service.

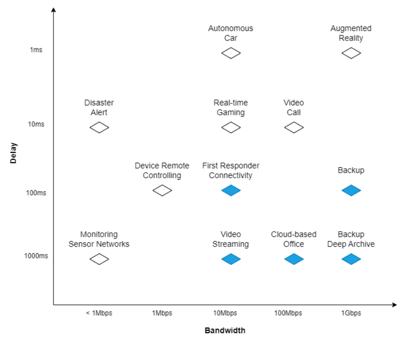


Figure 2: Comparison of QoS requirements (bandwidth and delay) for different cloud services.

Therefore, a balance between cost and latency is necessary for cost-benefit in the pay-per-use idea. Figure 2 reinforces our argument. Many services are delay tolerant as long as there is sufficient bandwidth. In this way, applications can be placed in many services, and their services could be placed in different cheaper regions that are further away from the customers so that the quality of service would not be degraded. This distribution of the application at different points of the infrastructure is possible based on composite applications. As we can see in Figure 1, software models and architectures that separate frontends and backends into distinct elements of the same application have benefited from the multi-region environment of cloud players. Applications developed under frameworks such as model-view-controler (MVC) (Mufid et al., 2019) or model-view-viewmodel (MVVM) (Sheikh and Sheikh, 2020), and more recently Microservices, can be hosted in physically distributed locations but still be logically composed.

3 RELATED WORK

Composite applications in cloud environments connect several smaller services into a larger and more complex application. The deployment of composite applications that maintain an excellent quality of service is a very complex action. Several works have been studying the problem of placement of decomposed applications in several services (Wen et al., 2017) (Mao et al., 2013). These composite applications can be deployed and operate in the cloud, taking advantage of the scalability, flexibility, and other advantages cloud computing offers. Application decoupling is a microservices design pattern that increases application scalability, availability, and efficiency is more resilient to failures and can be scaled more efficiently by splitting an application into minor services that can be deployed independently (Shi et al., 2020).

One offering an application over multi-region cloud environments also has the choice over which cloud player their application will be deployed (Warren et al., 2016). Generally, this choice falls on just one player, which offers the lowest price with acceptable latency. It is a good practice when the application is not composed of different entities. In composite applications, the choice of cloud player or players becomes more granular and can further improve the cost of application maintenance by distributing services across different cloud players. In situations like that, one of the main goals is finding the provisioning plan that incurs the smallest amount of different cloud players' resources so that the allocation cost is as low as possible. To handle such a scenario, The work (Faticanti et al., 2020) delivers an arrangement strategy that assigns microservice-based applications on federated cloud environments based on topological sorting techniques that determine the order in which microservices are allocated by their function in their application's workflow.

For administrators offering their applications, choosing the right place to host them is essential to balancing allocation cost and application performance. Whereas allocating applications closer to the customers guarantees low latency, high prices can be imposed, and places with low costs can impose high latencies. In such a scenario, the work (Mahmud et al., 2020) presents an allocation procedure that specifies whether applications should be hosted by the cloud resources to improve the profit while supplying acceptable performance levels for end-users. The work (Anglano et al., 2020) discusses an allocation scheme where cloud providers collaborate to maximize their profit by sharing resources to process time-varying workloads. The work (Li et al., 2022) gives a placement and scheduling framework, which leverages the collaboration between multiple cloud players to handle applications according to their deadline and allocation cost. The review of the papers mentioned above indicates a need to investigate the service deployment problem considering the response time of composite applications to accurately measure their real performance in multi-cloud.

4 SYSTEM MODEL

This section describes the composite application provisioning scenario approached in this work. First, we present the attributes and behaviors of the infrastructure's elements. Then, we detail our optimization objective. Table 1 summarizes the notations.

The network infrastructure comprises a set of geographically distributed regions, following the global infrastructure model employed by large-scale cloud providers such as Amazon Web Services. The network infrastructure is represented by an undirected graph $\mathcal{G} = (\mathcal{R}, \mathcal{L})$, where $\mathcal{R}$ represents the set of regions and $\mathcal{L}$ denotes the set of network links that interconnect the different regions. In this setting, a network link is modeled as $\mathcal{L}_f = \{g_f\}$, where g_f represents $\mathcal{L}_f$'s latency. To facilitate the computation of network operations, let us define three generic helper functions. The first function, $\lambda(\mathcal{E})$, encapsulates the Dijkstra algorithm (Dijkstra et al., 1959) for finding the shortest network path interconnecting a vector $\mathcal{E}$

Table 1: Summary of notations used in this paper.

Symbol	Description
$\mathcal{G}$	Network infrastructure
$\mathcal{R}$	Set of regions
$\mathcal{L}$	Set of network links
$\mathcal{P}$	Set of infrastructure providers
$\mathcal{D}$	Set of data centers
$\mathcal{A}$	Set of applications
$\mathcal{S}$	Set of services
$\mathcal{U}$	Set of users
g_f	$\mathcal{L}_f$'s latency
u_j	$\mathcal{A}_j$'s user
w_j	$\mathcal{A}_j$'s services
z_j	$\mathcal{A}_j$'s latency
s_j	$\mathcal{A}_j$'s latency SLA
n_j	$\mathcal{A}_j$'s communication path
h_k	$\mathcal{S}_k$'s demand
t_k	$\mathcal{S}_k$'s features specification
c_i	$\mathcal{D}_i$'s capacity
d_i	$\mathcal{D}_i$'s demand
r_i	$\mathcal{D}_i$'s region
$x_{i,k}$	Service placement matrix
$\lambda(\mathcal{E})$	Shortest path finder
$\psi(\mathcal{F})$	Path's accumulated delay
$\xi(E1,E2,T)$	Delay threshold compliance test
$\alpha(\mathcal{D}_i,\mathcal{S}_k)$	Cost for hosting $\mathcal{S}_k$ on $\mathcal{D}_i$
$\beta(\mathcal{A}_j)$	$\mathcal{A}_j$'s SLA compliance
$\phi(\mathcal{D}_i)$	$\mathcal{D}_i$'s capacity limit compliance

representing any two or more elements within the infrastructure (e.g., users, services, regions, etc.). The second function, $\psi(\mathcal{F})$, calculates the accumulated delay of any network path $\mathcal{F}$ found with $\lambda(\mathcal{E})$. The third function, $\xi(E1,E2,T)$ (Eq. 1), checks whether the delay between two elements ($E1$ and $E2$) is less than equal a given threshold T.

$$\xi(E1,\ E2,\ T) = \begin{cases} 1 & \text{if } \psi(\lambda([E1,\ E2])) \leqslant T \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

In our modeling, the computing infrastructure hosts a set of applications $\mathcal{A}$, which are accessed by a set of users $\mathcal{U}$, distributed across the various existing regions. An application is denoted as $\mathcal{A}_j = \{u_j, w_j, z_j, s_j, n_j\}$. This structure carries two attributes referencing other model components: u_j references $\mathcal{A}_j$'s user, and w_j references $\mathcal{A}_j$'s list of services. Complementarily, z_j represents $\mathcal{A}_j$'s latency incurred in the communication between the services in w_j, and s_j denotes $\mathcal{A}_j$'s latency Service Level Agreement (SLA), which indicates the maximum acceptable latency. The inter-service communication related to an application $\mathcal{A}_j$ is represented by a list n_j populated by the region where $\mathcal{A}_j$'s user is located and the regions where $\mathcal{A}_j$'s services are provisioned. The shortest network path connecting the elements in n_j is found through $\lambda(n_j)$ and z_j is calculated through $\psi(\lambda(n_j))$. In this context, $\beta(\mathcal{A}_j)$, described in Eq. 2, determines whether $\mathcal{A}_j$ experiences an SLA violation based on its SLA and its actual latency. Although this modeling has certain simplifications (e.g., each user accessing a single application, as in (Souza et al., 2022)), it can represent several provisioning scenarios since applications can be composed of single or multiple services with diverse communication patterns.

$$\beta(\mathcal{A}_j) = \begin{cases} 1 & \text{if } z_j > s_j \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The application provisioning consists of mapping a set of services $\mathcal{S}$ to a set of data centers $\mathcal{D}$ owned by a set of providers $\mathcal{P}$. We represent a service as $\mathcal{S}_k = \{h_k, t_k\}$, where h_k is the computational demand required to provision $\mathcal{S}_k$ and t_k is the specification of $\mathcal{S}_k$'s features. A data center is represented as $\mathcal{D}_i = \{c_i, d_i, r_i\}$, where attributes c_i, d_i, and r_i represent $\mathcal{D}_i$'s capacity, demand, and region, respectively. Based on this, the service placement is represented by $x_{i,k}$ (Eq. 3).

$$x_{i,k} = \begin{cases} 1 & \text{if data center } \mathcal{D}_i \text{ hosts } \mathcal{S}_k \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

As providers can define different cost policies according to the specifications of hosted services, $\alpha(\mathcal{D}_i,\mathcal{S}_k)$ represents the cost to host a service $\mathcal{S}_k$ in a data center $\mathcal{D}_i$. In addition, as data centers have limited capacity, the helper function $\phi(\mathcal{D}_i)$, described in Eq. 4, checks whether a given placement scheme respects the capacity constraints of a data center $\mathcal{D}_i$ (i.e., if services are not provisioned in such a way that $\mathcal{D}_i$'s demand does not exceed its capacity).

$$\phi(\mathcal{D}_i) = \begin{cases} 1 & \text{if } \sum_{k=1}^{|\mathcal{S}|} h_k \cdot x_{i,k} > c_i \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

We consider a twofold objective function (Eq. 5), which minimizes the number of SLA violations of applications in $\mathcal{A}$ while reducing the allocation cost of services in $\mathcal{S}$, subject to two constraints. While the first constraint (Eq. 6) ensures that each service is provisioned only in one data center, the second constraint (Eq. 7) ensures that the data centers' demands do not exceed their capacity.

$$\textbf{Min: } norm\left(\sum_{j=1}^{|\mathcal{A}|} \beta(\mathcal{A}_j)\right) + norm\left(\sum_{i=1}^{|\mathcal{D}|}\sum_{k=1}^{|\mathcal{S}|} \alpha(\mathcal{D}_i,\mathcal{S}_k)\cdot x_{i,k}\right) \tag{5}$$

Subject to:

$$\sum_{i=1}^{|\mathcal{D}|} x_{i,k} \ = \ 1, \forall k \in \{1,\ldots,|\mathcal{S}|\} \tag{6}$$

$$\phi(\mathcal{D}_i) \ = \ 0, \forall i \in \{1, ..., |\mathcal{D}|\} \tag{7}$$

5 CLOVER DESIGN

This section presents *Clover*, an algorithm that performs cost and latency optimized decisions while provisioning composite applications in geo-distributed data centers.

Algorithm 1: Clover algorithm.

```
1  A' = Applications sorted by Eq. 8 (desc.)
2  foreach A_j ∈ A' do
3      foreach S_k ∈ w_j do
4          D' = Data centers sorted by Eq. 15
           (desc.)
5          foreach D_i ∈ D' do
6              if c_i ≥ d_i + h_k then
7                  Provision S_k on D_i
8                  break
9              end
10         end
11     end
12 end
```

Clover employs a depth mapping approach, provisioning all the services of a given application before moving on to the services owned by the others. Consequently, one of *Clover*'s main decisions is to define the application provisioning order since a careless application arrangement might cause applications to be improperly provisioned, occupying resources that others could better use. *Clover* employs a score function (Eq. 8) that defines the application provisioning order based on two factors: (i) the network proximity of an application's user to the data centers and (ii) the potential allocation cost reduction achieved by prioritizing an application's provisioning (Alg. 1, line 1).

$$\partial(\mathcal{A}_j) = norm(\rho(\mathcal{A}_j)) + norm(\theta(\mathcal{A}_j)) \tag{8}$$

The first application sorting criterion (Eq. 9) aims to reduce SLA violations. For this, applications with fewer nearby resources are prioritized since postponing their provisioning increases the chances of other applications using those resources, forcing them to be provisioned in distant data centers that would violate the SLAs. Although simply counting the number of data centers whose latency does not violate the SLAs could work in homogeneous infrastructures, geo-distributed environments can comprise data centers with different capacities—as such, solely looking at the number of nearby data centers could bring misleading insights. Therefore, *Clover*'s sorting criterion evaluates the amount of resources close enough to the application's user to couple with the application's latency SLA.

$$\rho(\mathcal{A}_j) = \frac{1}{\sum_{i=1}^{|\mathcal{D}|}(c_i - d_i) \cdot \xi(r_i, u_j, s_j)} \tag{9}$$

The second application sorting criterion (Eq. 10) aims to reduce the applications' allocation cost. First, *Clover* prioritizes applications composed of services whose allocation costs vary significantly from one data center to another (Eq. 11), as prioritizing them can lead to higher cost savings than applications whose allocation costs are similar for all data centers. In addition, this sorting criterion considers the number of data centers with the lowest cost (Eq. 12) since the smaller the number of cheap data centers, the greater the risk of losing potential profit if other applications use their resources.

$$\theta(\mathcal{A}_j) = \sum_{\mathcal{S}_k \in w_j} \nu(\mathcal{S}_k) \cdot \mu(\mathcal{S}_k) \tag{10}$$

$$\nu(\mathcal{S}_k) = max(1,\ max(\gamma(\mathcal{S}_k)) - min(\gamma(\mathcal{S}_k))) \tag{11}$$

$$\mu(\mathcal{S}_k) = \frac{1}{|[b \mid b \in \gamma(\mathcal{S}_k) \land \chi(b, \gamma(\mathcal{S}_k)) = 1]| \cdot min(\gamma(\mathcal{S}_k))} \tag{12}$$

$$\chi(A,B) = \begin{cases} 1 & \text{if } A = min(B) \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

$$\gamma(\mathcal{S}_k) = \{\alpha(\mathcal{D}_i, \mathcal{S}_k) \mid \mathcal{D}_i \in \mathcal{D}\} \tag{14}$$

After sorting applications, *Clover* iterates over the list of services of each application to host them within the infrastructure (Alg. 1, lines 3–10). At this point, *Clover* employs a score function (Eq. 15) that ranks candidate data centers to host each service (Alg. 1, line 4). *Clover*'s data center sorting approach is based on two criteria: (i) whether the data center is close enough to the application's user and any previously provisioned application service to respect the application SLA, and (ii) the cost charged by the data center to host the service. *Clover* assumes that data centers cannot host services whose demand exceeds their capacity. Therefore, when iterating over the sorted list of candidate data centers to host a given service, it first checks whether it has sufficient available capacity to host the service before starting the provisioning procedure (Alg. 1, line 6).

$$\aleph(\mathcal{D}_i, \mathcal{A}_j, \mathcal{S}_k) = \xi(r_i, w_{j,k-1}, s_j) + norm\left(\frac{1}{\alpha(\mathcal{D}_i, \mathcal{S}_k)}\right) \tag{15}$$

6 PERFORMANCE EVALUATION

This section details the experiments conducted to assess *Clover* while provisioning composite applications on geo-distributed cloud data centers. First, we describe our research methodology (§6.1). Then, we present a sensitivity analysis that calibrates the configurable parameters of NSGA-II—one of the baseline strategies (§6.2). Finally, we discuss the obtained results (§6.3).

6.1 Experiments Description

We consider a geo-distributed infrastructure composed of nine regions interconnected by network links whose latency values are specified in Figure 3. Latency values are provided by CloudPing[1] and represent the 50th percentile among annual observations of latency between Amazon Web Services regions.

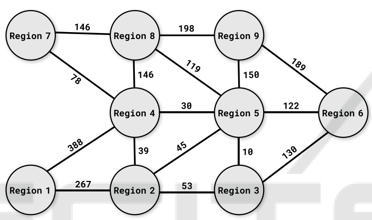


Figure 3: Network Latency between regions.

In our dataset, three infrastructure providers manage six data centers each. Data centers are evenly positioned across the nine regions, with capacity = {60, 120}—we consider such abstract units to represent data center capacities as real values could not be found publicly. Data centers must host 60 composite applications totaling 132 services. The sample application workflow specifications used in the experiments are presented in Figure 4. In this setting, applications have three distinct SLAs = {100, 150, 200}, and services have heterogeneous demands = {2, 4, 6, 8}, both uniformly distributed.

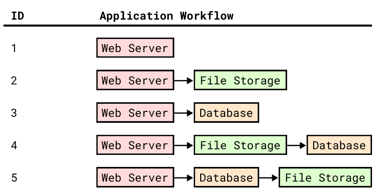


Figure 4: Application workflows used in the evaluation.

We compare *Clover* against three approaches. The first two baseline strategies, Best Fit and Worst Fit, allow us to evaluate the impact of extreme provisioning decisions on the infrastructure. While Best Fit focuses on consolidating services on a subset of data centers, Worst Fit does the opposite, distributing services across the infrastructure as much as possible. As both Best Fit and Worst Fit are naive strategies that make no optimized decision for the addressed scenario, our evaluation also considers a metaheuristic called Non-Dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002), which employs a population-based search procedure to find Pareto-optimal solutions. We chose NSGA-II over other algorithms that find Pareto-optimal solutions as its efficiency has been shown in many multi-objective problems (Deb et al., 2002; Aral et al., 2021). We compare the analyzed strategies regarding latency SLA violations and allocation cost (see Section 4 for details). Our simulations are primarily based on features provided by NetworkX (Hagberg et al., 2008), a well-known Python library that allows modeling large-scale network topologies. Our research strives to follow open science principles. Therefore, companion material publicly available on GitHub[2] contains the dataset, source code, and instructions for reproducing our results.

6.2 NSGA-II's Sensitivity Analysis

We conducted a sensitivity analysis to determine the best parameters for the NSGA-II algorithm. Without loss of generality, we define the population size as 400 and crossover probability to 100%. The algorithm was configured with the Uniform Crossover and Polynomial Mutation methods (Blank and Deb, 2020). The initial population was set using a Random-Fit Algorithm (Souza et al., 2022). Our analysis evaluates different population sizes = {100, 200, 300, ..., 1500} and mutation probabilities = {0%, 10%, 20%, 30%, ..., 100%}. Each parameter combination ω was accessed by a cost function Δ (Eq. 16), which considers both the normalized number of SLA violations and the allocation cost achieved by the solutions. In total, we evaluated 165 parameter combinations. As indicated in Figure 5, the NSGA-II algorithm achieved the lowest cost Δ with 1200 generations and mutation probability = 10%—accordingly, we used this configuration in the evaluation of Section 6.3.

$$\Delta(\omega) = norm\left(\omega^{SLAViolations}\right) + norm\left(\omega^{allocationCost}\right) \quad (16)$$

[1]https://www.cloudping.co/

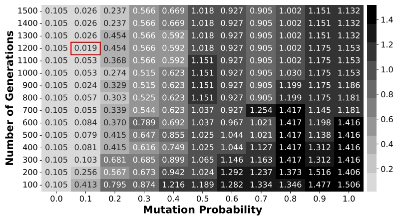


Figure 5: NSGA-II sensitivity analysis results.

6.3 Results and Discussion

6.3.1 Latency SLA Violations

Figures 6 (a) and (b) present the number of SLA violations during the execution of the evaluated placement strategies. As expected, the naive algorithms (Worst Fit and Best Fit) showed the worst results, with roughly 4x more SLA violations than *Clover* and NSGA-II. Although Best Fit and Worst Fit do not incorporate SLA-optimized allocations, we can make some observations about the impact of certain resource management decisions based on such results. As shown in Figure 6 (a), Best Fit's consolidation led to better outcomes than Worst Fit's spread strategy, especially for applications with larger service sets. While demand distribution brings several benefits (e.g., high availability and mitigation of saturations at specific points in the infrastructure), if not accompanied by SLA-aware policies, it can significantly degrade users' experience due to excessively long response times, which nullify any other potential benefits. As the number of SLA violations is computed per application, preventing violations from applications with fewer services is easier since they naturally have lower overall demand than applications with more services. That being said, while Worst Fit and Best Fit had a balanced number of SLA violations regardless of the application size (i.e., number of services), we can observe that *Clover* benefited from this, achieving results closer to NSGA-II's Pareto -optimal outcome. Figure 6 (a) shows that *Clover* could reduce the number of SLA violations of smaller applications, completely mitigating the violations from applications with a single service. In addition, Figure 6 (b) shows that *Clover* provides more steady latency for applications, reaching outcomes close to NSGA-II's, unlike the naive algorithms, which displayed a high standard deviation.

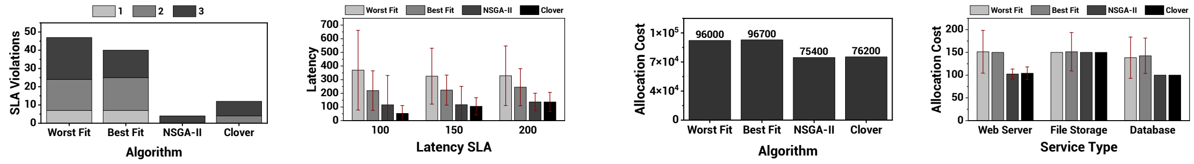
6.3.2 Allocation Cost

Figures 6 (c) and (d) present the application cost allocation results. Like the SLA violation analysis, Worst Fit and Best Fit delivered the worst outcomes. Unlike *Clover* and NSGA-II, Best Fit and Worst Fit exhibited a high standard deviation related to allocation cost per service, as they overlook this target metric (Figure 6 (d)). Best Fit's slightly higher overall cost allocation probably comes from its consolidation approach. If Worst Fit decides to allocate a service in an expensive data center, chances are that it will not use that data center excessively, given its demand-spreading approach. Conversely, a careless decision in terms of allocation cost performed by Best Fit can cause a more significant impact, as it tends to host as many services as possible in the same data center. As shown in Figure 6 (c), *Clover*'s provisioning decisions resulted in an allocation cost increase of only 1.06% compared to NSGA-II's Pareto-optimal outcome. This positive result derives primarily from two resource management decisions. First, *Clover* prioritizes applications with greater cost-saving potential and with a smaller number of cheap data centers. Consequently, it avoids wasting (potentially scarce) cheap resources with applications whose probable cost reduction is negligible. Second, *Clover*'s data center sorting scheme prioritizes hosting services in data centers with the lowest allocation cost as long as they don't meet latency SLAs.

7 CONCLUSION AND FUTURE WORK

A software architecture that allows decomposing the application into pieces of software, such as microservices, consists of an organization that is very well suited to the context of cloud computing. It occurs because using multiple cloud providers in several locations, especially when deploying composite applications like microservices, can deliver many cost benefits. First, it enables excellent flexibility in selecting cost-effective alternatives for specific microservices or geographic regions. Each microservice may have distinct resource requirements and performance needs, so using different providers for different microservices can be more cost-efficient. Then, using multiple cloud providers in different regions can also help optimize data transfer and reduce egress costs, which can be a high cost when transferring data between different regions. This paper analyzes the latency-aware cost-efficient composite application placement problem, introducing a novel model that captures these objectives during the provisioning procedure. To tackle this challenging scenario, we present *Clover*, a novel algorithm that leverages score-based functions to optimize the placement of

[2]https://github.com/paulosevero/clover



(a) SLA Violations per Chain Size.
(b) Latency per SLA.
(c) Overall Allocation Cost.
(d) Allocation Cost per Service Type.

Figure 6: Experimental results comparing the evaluated strategies.

composite applications in geo-distributed cloud data centers. Simulated experiments using real latency traces from Amazon Web Services demonstrate that *Clover* can achieve near-optimal results, reducing latency issues and placement costs by 74.47% and 21.2%, respectively, compared to baseline strategies. As future work, we indent to extend our approach, considering other objectives such as high availability, protecting applications against unfortunate events such as cyberattacks and power outages, and mitigating vendor lock-in.

ACKNOWLEDGEMENT

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Finance Code 001. Also, this work was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq – 313111/2019-7, 404027/2021-0. This work also received funding from São Paulo Research Foundation (FAPESP) – 2021/06981-0, 2020/05183-0, 2020/05115-4; and Rio Grande do Sul Research Foundation (FAPERGS) – 19/2551-0001266-7, 19/2551-0001224-1, 19/2551-0001689-1, 21/2551-0000688-9.

REFERENCES

Anglano, C., Canonico, M., Castagno, P., Guazzone, M., and Sereno, M. (2020). Profit-aware coalition formation in fog computing providers: A game-theoretic approach. *Concurrency and Computation: Practice and Experience*, 32(21):e5220. e5220 cpe.5220.

Aral, A., Demaio, V., and Brandic, I. (2021). Ares: Reliable and sustainable edge provisioning for wireless sensor networks. *IEEE Transactions on Sustainable Computing*, pages 1–12.

Blank, J. and Deb, K. (2020). Pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509.

Blinowski, G., Ojdowska, A., and Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, 10:20357–20374.

Chaudhari, N., Bhadoria, R. S., and Prasad, S. (2016). Information handling and processing using enterprise service bus in service-oriented architecture system. In *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 418–421.

De Donno, M., Tange, K., and Dragoni, N. (2019). Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. *IEEE Access*, 7:150936–150948.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Dijkstra, E. W. et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Faticanti, F., Savi, M., Pellegrini, F. D., Kochovski, P., Stankovski, V., and Siracusa, D. (2020). Deployment of application microservices in multi-domain federated fog environments. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–6.

Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Ilie, L., Pop, E., Caramihai, S. I., and Moisescu, M. A. (2022). A soa based e-health services framework. In *2022 E-Health and Bioengineering Conference (EHB)*, pages 1–4.

Katsuno, Y. and Takahashi, H. (2015). An automated parallel approach for rapid deployment of composite application servers. In *2015 IEEE International Conference on Cloud Engineering*, pages 126–134.

Kozina, O. and Kozin, M. (2022). Simulation model of data consistency protocol for multicloud systems. In *2022 IEEE 3rd KhPI Week on Advanced Technology (KhPI-Week)*, pages 1–4.

Li, Y., Dai, W., Gan, X., Jin, H., Fu, L., Ma, H., and Wang, X. (2022). Cooperative service placement and scheduling in edge clouds: A deadline-driven approach. *IEEE Transactions on Mobile Computing*, 21(10):3519–3535.

Liu, J., Qiao, J., and Zhao, J. (2018). Femcra: Fine-grained elasticity measurement for cloud resources allocation. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 732–739.

Mahmud, R., Srirama, S. N., Ramamohanarao, K., and Buyya, R. (2020). Profit-aware application placement for integrated fog–cloud computing environments. *Journal of Parallel and Distributed Computing*, 135:177–190.

Mao, Z., Yang, J., Shang, Y., Liu, C., and Chen, J. (2013). A game theory of cloud service deployment. In *2013 IEEE Ninth World Congress on Services*, pages 436–443.

Mufid, M. R., Basofi, A., Al Rasyid, M. U. H., Rochimansyah, I. F., and rokhim, A. (2019). Design an mvc model using python for flask framework development. In *2019 International Electronics Symposium (IES)*, pages 214–219.

Pelle, I., Czentye, J., Dóka, J., and Sonkoly, B. (2019). Towards latency sensitive cloud native applications: A performance study on aws. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 272–280.

Raj, P., Vanga, S., and Chaudhary, A. (2023). *Design, Development, and Deployment of Event&x2010;Driven Microservices Practically*, pages 129–142.

Sheikh, W. and Sheikh, N. (2020). A model-view-viewmodel (mvvm) application framework for hearing impairment diagnosis - type dependency architecture. In *2020 Intermountain Engineering, Technology and Computing (IETC)*, pages 1–6.

Shi, T., Ma, H., Chen, G., and Hartmann, S. (2020). Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment. *IEEE Transactions on Parallel and Distributed Systems*, 31(8):1954–1969.

Souza, P., Ângelo Crestani, Rubin, F., Ferreto, T., and Rossi, F. (2022). Latency-aware privacy-preserving service migration in federated edges. In *International Conference on Cloud Computing and Services Science*, pages 288–295.

Vu, T., Mediran, C. J., and Peng, Y. (2019). Measurement and observation of cross-provider cross-region latency for cloud-based iot systems. In *2019 IEEE World Congress on Services (SERVICES)*, volume 2642-939X, pages 364–365.

Warren, M. S., Skillman, S. W., Chartrand, R., Kelton, T., Keisler, R., Raleigh, D., and Turk, M. (2016). Data-intensive supercomputing in the cloud: Global analytics for satellite imagery. In *2016 Seventh International Workshop on Data-Intensive Computing in the Clouds (DataCloud)*, pages 24–31.

Wen, Z., Cała, J., Watson, P., and Romanovsky, A. (2017). Cost effective, reliable and secure workflow deployment over federated clouds. *IEEE Transactions on Services Computing*, 10(6):929–941.