sensors

MDPI

Article

LM^2K Model for Hosting an Application Based on Microservices in Multi-Cloud

Juliana Carvalho [1], Dario Vieira [2,*], Christiano Rodrigues [2,3] and Fernando Trinta [3]

[1] Information System Department, Federal University Piauí—(UFPI), Picos 64607-670, PI, Brazil; julianaoc@ufpi.edu.br
[2] Efrei Research Lab, EFREI Paris, 94800 Villejuif, France
[3] Computing Department, Federal University Ceará—(UFC), Fortaleza 60440-900, CE, Brazil; fernando.trinta@dc.ufc.br
* Correspondence: dario.vieira@efrei.fr

Abstract: Cloud computing has become a popular delivery model service, offering several advantages. However, there are still challenges that need to be addressed when applying the cloud model to specific scenarios. Two of such challenges involve deploying and executing applications across multiple providers, each comprising several services with similar functionalities and different capabilities. Therefore, dealing with application distributions across various providers can be a complex task for a software architect due to the differing characteristics of the application components. Some works have proposed solutions to address the challenges discussed here, but most of them focus on service providers. To facilitate the decision-making process of software architects, we previously presented PacificClouds, an architecture for managing the deployment and execution of applications based on microservices and distributed in a multi-cloud environment. Therefore, in this work, we focus on the challenges of selecting multiple clouds for PacificClouds and choosing providers that best meet the microservices and software architect requirements. We propose a selection model and three approaches to address various scenarios. We evaluate the performance of the approaches and conduct a comparative analysis of them. The results demonstrate their feasibility regarding performance.

Keywords: multi-cloud; microservice; cloud selection

Citation: Carvalho, J.; Vieira, D.; Rodrigues, C; Trinta, F. LM^2K Model for Hosting an Application Based on Microservices in Multi-Cloud. *Sensors* **2023**, *23*, 4450. https://doi.org/10.3390/s23094450

Academic Editor: Seongsoo Cho

Received: 7 February 2023
Revised: 7 April 2023
Accepted: 13 April 2023
Published: 2 May 2023

1. Introduction

Cloud computing has emerged as a popular delivery model service in recent years. According to [1], it provides a large set of easily accessible and usable virtualized resources. We can dynamically reconfigure these resources to adapt to variable loads, enabling optimized resource utilization. Cloud providers typically use a pay-per-use model and must offer infrastructure guarantees through customized service-level agreements.

Applications built on cloud concepts offer benefits such as elastic growth, easy maintenance and updating, and efficient resource utilization through multi-tenancy approaches. However, building cloud-centric applications presents several challenges from a software engineering perspective. These obstacles encompass issues such as data privacy, migrating applications to the cloud, and vendor lock-in, a problem where applications are restricted to specific providers [2,3]

In recent years, several works have proposed solutions to the challenges related to developing native cloud applications, including those with system components distributed across multi-cloud providers. The distribution of these components considers requirements such as performance, availability, and operational costs of the service from the provider [4–6]. This scenario, known as a multi-cloud provider environment, provides software architects with various possibilities to set up applications while ensuring the service that best meets the application needs, regardless of which provider offers the service [7,8].

A multi-cloud provider environment encompasses different views on how applications use them. In this sense, different works proposed taxonomies and classifications, such as [8–10]. According to [11], there are three cloud service delivery models: multi-cloud, cloud federation, and inter-cloud. In this work, we adopted the multi-cloud service delivery model, as it does not depend on an agreement between providers and, consequently, offers a higher number of providers and cloud services.

Using multi-cloud providers can bring many economic benefits, especially if we consider the significant growth in the number of cloud providers and the number of resources they offer. However, many challenges remain, such as security, privacy, trust, legal issues, resource management, and the service-level agreement (SLA) of services [9]. In order to address these problems, we proposed PacificClouds [12], an approach that aims to manage the entire process of deploying and executing a microservice-based application distributed in a multi-cloud environment from the software architect's perspective.

A microservice is an architectural style that has stood out in the design of cloud applications. According to [13], microservices decompose a monolithic business system into independently deployable services. Thus, we can build applications and organize codes in several different parts. Moreover, we execute them in separate processes, as described by [14]. The suitability of this architectural style for cloud applications also makes it a natural candidate for applications distributed in a multi-cloud environment. Therefore, in this paper, we focus on applications based on microservices.

In this paper, we propose a new provider selection model and three different approaches for this model. Unlike our previous works [15–17], we now consider that the communication time between microservices deployed in different providers must be handled. Next, we detail our main contributions.

1. We propose a new provider selection model named Link Microservice Mapped to the Knapsack (LM^2K).
2. We propose and implement three approaches for this model: (i) dynamic, (ii) greedy, and (iii) ant colony.
3. Unlike other works in the literature, our proposed model considers that a microservice can only be hosted by a single provider, decreasing the communication costs among cloud services.
4. We evaluate the performances of all approaches, compare the results obtained, and present each approach's feasibility.
5. We expand the taxonomy provider and service granularity (PSG) [17]. It seeks to identify the number of providers and services before and after selecting providers for the models and approaches proposed in the literature.

The remainder of this paper is organized as follows. Section 2 presents the multi-cloud providers scenario and the vendor lock-in problem. Next, we present related works in Section 3. In Sections 4 and 5, we present the LM^2K model, the three proposed approaches, and their implementations. In Section 6, we describe the tools, scenarios, experiments, and discuss the results. Finally, in Section 7, we present our conclusions and future works.

2. Multiple Cloud Providers

Cloud computing proliferation occurs mainly due to its essential characteristics: resource pooling, broad Internet access, measured service, on-demand self-service, and rapid elasticity [18]. Moreover, cloud providers present the resources as if they are inexhaustible. In this model, users pay cloud providers based on resource usage and do not worry about infrastructure and service management, reducing operating costs for enterprises.

In this context, many cloud providers have emerged. In order to serve the largest number of customers and win them over, many providers have adopted specific strategies in how their users use resources. For this reason, users who want to make the most of cloud computing, either by using services from multiple providers to compose more complex and innovative applications, or by migrating from one provider to another to obtain better

quality services, are required to rework their applications. In this way, the user becomes hostage to the cloud provider; this is also known as the [2] vendor lock-in problem.

One way to take advantage of cloud computing is to use multiple clouds. According to [11,19], a multi-cloud application uses geographically distinct cloud resources, sequentially or simultaneously. There are several terminologies for multi-cloud environments. Following our previous works, we adopt the definition presented in [11] and, therefore, we classify the delivery model used in this work as multi-cloud.

The resource management functions in multiple clouds are major challenges; they are responsible for (i) selecting resources in various clouds for the application deployment, (ii) monitoring the execution time of the application resources distributed across multiple clouds according to the functional and non-functional requirements, and (iii) monitoring the cloud providers' resources. Resource management in multiple clouds can be thought of from the perspectives of software architects and providers. In this context, resource management plays an essential role in the distribution and execution of a distributed application and, consequently, in interoperability and portability. It makes the multi-cloud environment more flexible by offering the most appropriate resources to meet the requirements, both in the deployment and in the execution of an application.

3. Related Work

In this work, we propose the LM^2K model to address the multi-cloud provider's selection to deploy microservices from a software architect's perspective. Furthermore, we propose three solutions: (i) a dynamic algorithm, (ii) a greedy algorithm, and (iii) an ant colony-based algorithm.

Other works in the literature also deal with the cloud provider selection problem. However, due to the number of open issues in this subject, the primary concern of these works is scattered across different aspects. This section describes and compares some research works that address the cloud provider selection problem from the software architect's perspective and divides the works into three categories. The first category describes works that select services from a single cloud provider. The second presents works that select only one service among the services offered by multiple providers. Finally, in the third category, the works select various services from multiple providers.

3.1. Single Cloud for Service Selection

Reference [20] proposed a service composition using the eagle strategy with whale optimization algorithm (ESWOA). In this work, the authors used simple additive weighting (SAW) to classify candidate services, and user-defined requirement thresholds. The proposed solution only uses a sequential service composition structure but transforms the other types of structures into sequential structures. [6] presented the optimal fitness-aware cloud service composition (OFASC) using an adaptive genetic algorithm based on genotype evolution (AGEGA), which considers various quality of service (QoS) parameters. It provides solutions that satisfy the QoS parameters and the connectivity restrictions of the service composition.

Reference [21] proposed a structure for cloud service selections with criteria interactions (CSSCI) that applied a fuzzy measure and a Choquet integral to measure and aggregate non-linear relations between criteria. The authors used a non-linear constraint optimization model to estimate the interaction ratios of importance and Shapley's criteria.

Reference [22] introduced a modified particle swarm optimization (PSO) approach based on QoS that reduces the search space for composing cloud services. They also proposed a modified PSO-based cloud service composition algorithm (MPSO-CSC), which removes the dominant cloud services and then employs the PSO to find the ideal cloud service set.

Reference [23] deals with the selection and dynamic composition of services for multiple tenants. The authors proposed a multi-tenant middleware for dynamic service composi-

tion in the Software-as-a-Service (SaaS) cloud. They presented a coding representation and adequacy functions to model service selection and composition as an evolutionary search.

3.2. Multiple Clouds for a Service Selection

The works described in this subsection used multi-cloud providers to select only one service from one cloud provider. Thus, despite using multi-cloud providers in the selection process, they did not select multiple services from multi-cloud providers or interrelate services and providers as in this work.

Reference [24] presented a classification-oriented prediction method that helps in the process of discovering candidate cloud services that generate greater user satisfaction. The approach proposed by the authors encompasses two primary functions: service classification by similarity and service classification prediction in the cloud, taking into account a user's preference. For this, the authors used a cloud service classification prediction method called CSRP, which covers the identification of similar neighbors, customer preferences, customer satisfaction estimation, and classification prediction.

Reference [25] proposed a hybrid multi-criteria decision-making model for selecting services from the available providers. The methodology assigns several classifications for cloud services based on QoS parameters, using an extended gray technique for order preference by similarity to ideal solution (TOPSIS), integrated with the hierarchical analytical process (AHP).

3.3. Multiple Clouds for Multiple Service Selections

This subsection describes works that used multiple cloud providers in the selection process, and they selected multiple services from multiple providers. The works presented in this subsection selected only one service from each provider, while our models select microservices that require several cloud services.

Reference [26] proposed two task-scheduling algorithms in a cloud federation environment, both based on a common SLA. The algorithm receives a set of independent tasks and a set of m cloud providers, along with the runtime, gain, and cost penalty of tasks at different cloud providers. The problem is to schedule all tasks to the clouds concerning the SLA so that there is a balance between the overall processing time and the cost gain penalty.

Reference [4] proposed an automated approach for the selection and configuration of cloud providers for a microservice in a multi-cloud environment. The approach proposed by the authors uses domain-specific language to describe the requirements of an application's multi-cloud environment and provides a systematic method for obtaining appropriate configurations that meet the requirements of an application and the restrictions of cloud providers. The solution proposed by [4] deals with cloud service selections composing microservices, in which the cloud services must be different cloud providers.

Reference [27] proposed a composite service selection (CSS) approach to address the configuration that involves multiple simultaneous requests for the composite service. The authors selected various service combinations in a multi-cloud environment. However, in our model, the services of the same combination can be in different cloud providers. Additionally, their work only deals with the sequential structure in each combination and does not consider the interactions among the selected combinations.

Reference [28] proposed a multi-objective hybrid evolutionary algorithm called ADE-NSGA-II to compose services in an inter-cloud environment, meeting users' QoS requirements. In the proposed algorithm, the authors used adaptive mutation and a crossover operator to replace strategies of the genetic algorithm of non-dominated ordering-II (NSGA-II). The work of [28] differs from ours, as they dealt with the composition of services when selecting a service from each provider.

Table 1 summarizes the main characteristics of all related works presented in this section. The first two columns of the table enumerate and reference the related work. The third, fourth, and fifth columns use a taxonomy to identify the focus of the work concerning

the number of providers and services used in each phase, which we call provider and service granularity (PSG). The third column shows the number of providers in the selection process, the fourth shows the number of selected providers, and the fifth indicates the granularity of services per selected provider. Column six presents the methods used to solve the selection problem, and the last column defines whether the user defines the requirements threshold. Furthermore, in Table 1, $\mathtt{n}$ indicates the number of providers in the selection process ($\mathtt{n} > 1$), and $\mathtt{m}$ indicates the number of providers selected by the selection process ($1 < \mathtt{m} \leq \mathtt{n}$).

Table 1. Summary of related work characteristics.

Items	Related Work	Characteristics				
		PSG			Method	User-Defined Threshold
		(1)	(2)	(3)		
1	[20]	1	1	Service Composition	ESWOA	Yes
2	[6]	1	1	Service Composition	OFASC and AGEGA	No
3	[21]	1	1	Service	Authors-defined	No
4	[22]	1	1	Service Composition	MPSO-CSC	Yes
5	[23]	1	1	Service Composition	Authors-defined	Yes
6	[24]	n	1	Service	CSRP	No
7	[25]	n	1	Service	Grey Technique, TOPSIS, and AHP	No
8	[26]	n	m	Tasks	Authors-defined	No
9	[4]	n	m	Service	Authors-defined	No
10	[27]	n	m	Service	Authors-defined	No
11	[28]	n	m	Service	Authors-defined	Yes
12	Dynamic UM^2K [15]	n	m	Microservice	SAW, Dynamic Algorithm	Yes
13	Greedy UM^2K [16]	n	m	Microservice	SAW, Greedy Algorithm	Yes
14	UM^2Q [17]	n	m	Microservice	SAW, Authors-defined	Yes
15	Dynamic LM^2K	n	m	Microservice	SAW, Dynamic Algorithm	Yes
16	Greedy LM^2K	n	m	Microservice	SAW, Greedy Algorithm	Yes
17	Ant Colony LM^2K	n	m	Microservice	SAW, Ant Colony Optimization Algorithm	Yes

PSG—providers and services granularity, (1) provider granularity in the selection process, (2) provider granularity in the selection result, (3) service granularity per provider.

4. Multi-Cloud Selection Model: LM^2K

PacicClouds uses various functionalities to meet different goals, and one of its primary tasks is to select the available cloud providers that best serve application microservices. The main objective of this work is to choose cloud providers for hosting the application microservices. However, many cloud providers are available, and each provider offers several services. Furthermore, an application can have multiple microservices, each of which may require several cloud services to meet its needs. Therefore, selecting suitable cloud providers to host an application distributed in multiple clouds is a complex task.

This section proposes a new multi-cloud selection model for PacificClouds to host applications based on microservices, named Linked Microservice Mapped to Knapsack (LM^2K). This model focuses on applications by considering communications among microservices and mapping the multi-choice knapsack problem.

The proposed selection model selects cloud providers from the software architect's perspective. Therefore, it is necessary to define their requirements before deploying an application. We assume that the provider's capabilities are known. In this work, we focus on three requirements: (i) response time, (ii) availability, and (iii) application execution cost. However, our model can support other requirements, such as energy cost and different types of cloud resources (e.g., CPU), as long as the cloud provider provides that information, and the architect considers it an important feature.

First, we will describe the formalization of the service model. Then, we will explain the process of selecting providers. All the descriptions of equation nomenclature are summarized in Appendix A.

4.1. Formalization

The required cloud services used to compose a microservice have different requirements, and the providers offer many services with the same functionalities but with different capabilities. In this work, we chose to assess the application response time (execution time plus delay), cloud availability, and application execution cost, but other requirements can be included in our model. We use three user requirements that are sufficient to help one understand the proposed selection process. However, several other user requirements can be defined without significant changes in the code. In addition, the requirements for the communication links among the microservices must be added to all requirements.

Definition 1. *Cloud service model —as S(S.rt,S.a,S.c), where S.rt is the response time, S.a is availability, and S.c is cost, as in [29].*

Definition 2. *Cloud service class—as* $SC_l = \{S_{l1}, S_{l2}, \ldots, S_{lo}\}$*, in which* S_{l1}, S_{l2}, …, S_{lo} *are services of the same provider with the same functionalities but different capabilities.*

Definition 3. *Service provider model—as* $SP_k = \{SC_{k1}, SC_{k2}, \ldots, SC_{kp}\}$*, in which* SC_{k1}, SC_{k2}, …, SC_{kp} *are service classes.*

Definition 4. *Cloud provider set—as* $CP = \{SP_1, SP_2, \ldots, SP_q\}$*, in which* SP_1, SP_2, …, SP_q *are service providers.*

Definition 5. *Microservice model—as* $MS_i = \{S_{i1}^{k_1}, S_{i2}^{k_2}, \ldots, S_{ir}^{k_r}\}$*, in which* $S_{i1}^{k_1}$, $S_{i2}^{k_2}$, …, $S_{ir}^{k_r}$ *are cloud services that are indispensable to executing a microservice, and they should be of different service classes. Thus,* $1 \leqslant k_r \leqslant o$*, in which* o *is the maximum number of classes for a cloud provider. Moreover, a microservice consists of a task flow, and cloud services must perform it. Figure 1 shows four basic structures to compose a task: sequential, circular, parallel, and selection [30]. In Figure 1,* X_v *indicates the* $S_{iv}^{k_v}$*, i.e., the service* j *from the provider* k *for the microservice* i*.*

This work considers the task flow for a microservice, as shown in Figure 2. The task flow is an acyclic graph, in which each vertex represents a service, and each arrow represents a precedence relation between the services, following the basic structures shown in Figure 1. All cloud services

used by a microservice must belong to the same provider. In Figure 2, X_v indicates the $S_{iv}^{k_v}$, i.e., the service j from the provider k for microservice i.

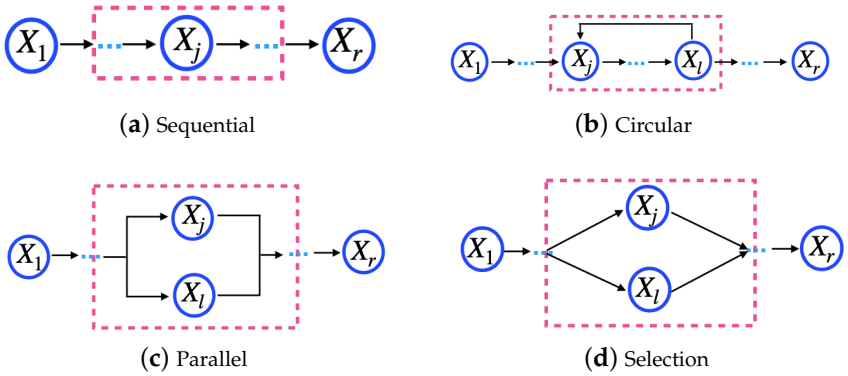

Figure 1. Structure of a microservice task.

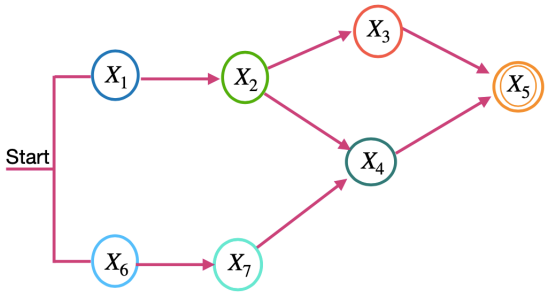

Figure 2. Task flow example.

Definition 6. *Application model—as $AP = \{MS_1, MS_2, \ldots, MS_t\}$, in which $MS_1, MS_2, \ldots, MS_t$ are microservices. For the application composition, microservices must use the basic structures shown in Figure 1. In Figure 1, X_v indicates the MS_v, i.e., microservice v.*

The application tasks are microservices. Therefore, an application task flow is a microservice flow. Figure 2 shows an example of a microservice flow considered in this work. In Figure 1, X_v indicates the MS_v, i.e., microservice v. It differs from a single microservice task flow because the microservices of an application can be hosted on different cloud providers. In contrast, the cloud services required for a microservice must belong to the same provider.

To calculate the microservice requirements in the LM^2K model, the microservice flow is divided into two levels: the term and the sequence. Figures 3 and 4 show the terms and sequences of the example of the microservice flow in Figure 2.

In Figure 3, terms 1 and 2 are the microservice flows of Figure 2, which must be performed in parallel. In Figure 4, the term sequences are the microservice flows for a term and only one of them is activated each time the microservice flow is executed.

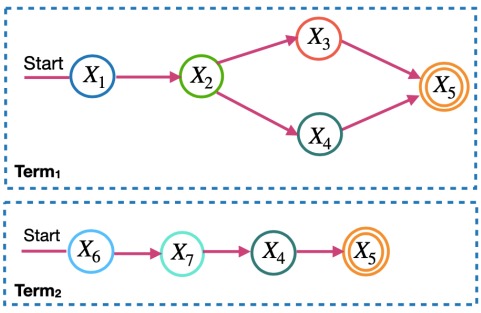

Figure 3. Terms of the microservice flow.

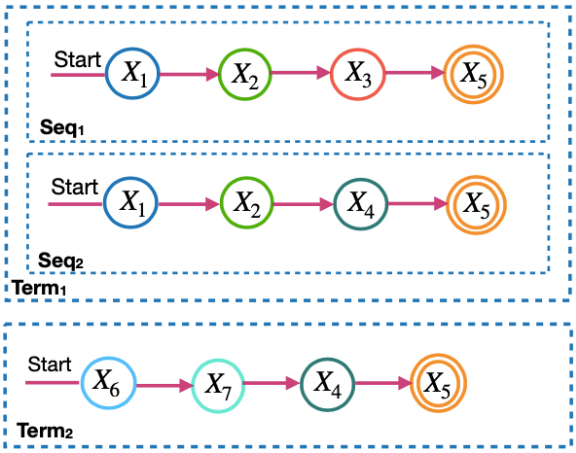

Figure 4. The sequences of the terms of the microservice flow.

The next subsections describe how to calculate the values required for application execution distributed across multi-cloud providers. For this, we consider the application microservice flow and communication links. We define communication links as the communication among microservices. The formulas for calculating the requirement values are in accordance with [30–34].

4.1.1. Availability Requirement

The cloud provider availability for an application must at least meet the threshold set up by a software architect. MinAvbty is the minimum availability threshold set by a software architect. In (1), we define AP.a as the availability, which is the product among all the application terms. Moreover, (2) defines $Term_j.a$ as the application term `j` availability. The availability of $Term_j.a$ is the sum of all sequence availabilities that belong to term `j`. In (3), we name $Seq_{ij}.a$ as the availability of the sequence `i` of the term `j`, which is the product of all microservice availabilities, the availability of all communication links, and the sequence selection probability represented by α (3).

$$AP.a = \prod_{j=1}^{num(Term)} Term_j.a \geqslant MinAvbty, \quad \forall Term_j \mid Term_j \in AP \tag{1}$$

$$Term_j.a = \sum_{i=1}^{num(Seq)} Seq_{ij}.a, \forall Seq_{ij} \mid Seq_{ij} \in Term_j \tag{2}$$

$$Seq_{ij}.a = \alpha \times \prod_{k=1}^{num(MS)} MS_k.a \times \prod_{k=1}^{num(MS-1)} Links_k.a, \quad \forall MS_k \mid MS_k \in Seq_{ij}, Link_k.ainSeq_{ij} \tag{3}$$

We calculate the availability of a microservice in a similar way to the application availability. The difference is that we based the application availability on the microservice flow. In contrast, we based the microservice availability on the cloud service availability. Cloud services are the services needed to compose a microservice.

4.1.2. Response Time Requirement

An application response time must meet the threshold set by a software architect. A software architect defines MaxRT as the maximum response time threshold, which is the sum of the maximum execution time threshold (MaxExecTime) and the maximum delay threshold (MaxDelay), as shown in (4). In (5), we define AP.rt as the response time of an application, which receives the longest response time of all terms in an application and must be less than or equal to MaxRT. Moreover, (6) defines $Term_j.rt$ as the response time

for the term `j`, which is assigned the sum of the response times for all strings that belong to the term `j`; (7) represents the response time for the sequence `i` of the term `j`; (7) represents the response time sum of all microservices that belong to the sequence, and the delays between each microservice multiplied by the sequence selection probability. We represent the sequence selection probability by α. We represent the delay between two microservices in a sequence of $Link_k.rt$.

$$MaxRT = MaxExecTime + MaxDelay \tag{4}$$

$$AP.rt = \max_{j=1}^{num(Term)} \{Term_j.rt\} \leqslant MaxRT, \quad \forall Term_j \mid Term_j \in AP \tag{5}$$

$$Term_j.rt = \sum_{i=1}^{num(Seq)} Seq_{ij}.rt, \forall Seq_{ij} \mid Seq_{ij} \in Term_j \tag{6}$$

$$Seq_{ij}.rt = \alpha \times \Big(\sum_{k=1}^{num(MS)} MS_k.rt + \sum_{k=1}^{num(MS-1)} Link_k.rt \Big), \quad \forall MS_k \mid MS_k \in Seq_{jj}, Link_k \in Seq_{ij} \tag{7}$$

We calculate the response time for a microservice in a similar way to the application's response time, but we based the application's response time on the microservice flow. In contrast, we based the microservice response time on the cloud service response time. Cloud services are the services required to compose a microservice.

4.1.3. Cost Requirements

The application execution cost should not exceed the budget, which is the cost threshold set up by a software architect. In (8), we define AP.c as the application execution cost, which is attributed to the total cost of all terms that belong to an application microservice flow and it must be less than or equal to the budget; (9) defines $Term_j.c$ as the term `j` cost, to which we assign the cost sum of all sequences belonging to the term `j`. We represent the sequence `i` cost from the term `j` in (10). Moreover, (10) receives the cost sum of all microservices that belong to the sequence and the communication links costs between each microservice multiplied by the sequence selection probability. We represent the sequence selection probability by α and the communication link cost between two microservices in a sequence by $Link_k.c$.

$$AP.c = \sum_{j=1}^{num(Term)} Term_j.c \leqslant Budget, \forall Term_j \mid Term_j \in AP \tag{8}$$

$$Term_j.c = \sum_{i=1}^{num(Seq)} Seq_{ij}.c, \forall Seq_{ij} \mid Seq_{ij} \in Term_j \tag{9}$$

$$Seq_{ij}.c = \alpha \times \Big(\sum_{k=1}^{num(MS)} MS_k.c + \sum_{k=1}^{num(MS-1)} Link_k.c \Big), \quad \forall MS_k \mid MS_k \in Seq_{ij}, Link_k \in Seq_{ij} \tag{10}$$

We calculate the availability, response time, and cost for a microservice in a similar way to the application availability, response time, and cost, respectively. The difference is that we based the application availability, response time, and cost on the microservice flow. In contrast, we based the microservice availability, response time, and cost on the cloud service availability, response time, and cost, respectively. The cloud services are the services required to compose a microservice.

4.2. Cloud Provider Selection Process

This subsection details the cloud provider selection process for the model, which is divided into three levels. The first level discovers all candidate cloud services of a microservice across all cloud providers. The second level determines the candidate providers for all microservices of an application. The third level selects providers to deploy all microservices of an application. As in our previous work [15–17], this process is a combinatorial optimization problem that we mapped to the multiple-choice knapsack problem. Next, the three levels are described in detail.

4.2.1. First Level

We select the services of each provider that meet all requirements of the software architects, which results in a set of candidate services for each provider. Next, we rank all candidate services in each provider. We use the simple additive weighting (SAW) technique as in [35], which has two phases: scaling and weighting.

- Scaling phase: First, a matrix $R = (R_{ij}; 1 \leqslant i \leqslant n; 1 \leqslant j \leqslant 3)$ is built by merging the requirement vectors of all candidate services. For this, the user requirements are numbered from 1 to 3, where 1 = availability, 2 = response time, and 3 = cost. The candidate services refer to the same microservice service. The entire process must be done for each microservice service. Each row R_i corresponds to a cloud service $S_{ij}^{k_j}$ and each column R_j corresponds to a requirement. Next, the requirements should be ranked using one of the two criteria described in Equations (11) and (12) in order to normalize the values. Equations (11) and (12) are used based on the requirements. For example, for the response time, the smaller, the better, contrary to availability, the greater, the better. Moreover, $R_j^{Max} = Max(R_{ij}), R_j^{Min} = Min(R_{ij}), 1 \leqslant i \leqslant n$.
  Negative: the higher the value ⇑, the lower the quality ⇓.

$$V_{ij} = \begin{cases} \frac{R_j^{Max} - R_{ij}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \tag{11}$$

  Positive: the higher the value ⇑, the higher the quality ⇑.

$$V_{ij} = \begin{cases} \frac{R_{ij} - R_j^{Min}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \tag{12}$$

- Weighting phase: All candidates will receive a weight, which is the overall requirements score, for each candidate cloud service (Equation (13)).

$$Score(S_i) = \sum_{j=1}^{3} (V_{ij} \times W_j) \mid W_j \in [0,1], \sum_{j=1}^{3} W_j = 1 \tag{13}$$

At the end of the first level, we must identify and classify all candidate services of a microservice for all available cloud providers.

4.2.2. Second Level

We must build all candidate cloud service combinations from a provider to compose a microservice. For this, we must combine candidate services from a microservice that is offered by the same provider; (14) defines SSP_{ik} as the set of candidate combinations from the provider `k` for the microservice `i`. In addition, in (14), $comb_{ikj}$ indicates the combination `j` from the provider `k` to the microservice `i`. In (15), a $comb_{ikj}$ combination is a set of cloud services from different classes to meet the microservice `i` needs. In (15), $S_{ij}^{k_i}$ indicates the candidate service `j` of the class `j` from the provider `k` needed for function `j` from microservice `i`. The microservice has a maximum of `r` functions and, consequently, requires

a maximum of r cloud services, a candidate provider has a maximum of m combinations and a microservice has a maximum of q candidate providers.

$$SSP_{ik} = \{comb_{ik1}, \ldots, comb_{ikm}\} \tag{14}$$

$$comb_{ikj} = \{S_{i1}^{k_1}, \ldots, S_{ir}^{k_r}\} \tag{15}$$

We must calculate each combination score and cost in SSP_{ik}, as shown in (16) and (17). Moreover, (16) defines $comb_{ikj}.score$ as the combination of each service score sum plus the communication link score sum between each service that belongs to the service flow of the microservice. Furthermore, (17) defines $comb_{ikj}.c$ as the total cost of all services plus the cost of each communication link between services in a combination, according to the service flow of the microservice.

$$comb_{ikj}.score = \sum_{x=1}^{r} Score(S_{ix}^{k_x}) + \sum_{x=1}^{r-1} Score(Link_x) \tag{16}$$

$$comb_{ikj}.c = \sum_{x=1}^{r} S_{ix}^{k_x}).c + \sum_{x=1}^{r-1} Link_x.c \tag{17}$$

At the end of the second level, there is a set of all candidate combinations from all providers available for application microservices (SAMS), and each combination has a score and a cost. Moreover, (18) defines SMS_i as a candidate providers' set for the microservice i. In (18), SSP_{ik} indicates the set of candidate combinations from the provider k to the microservice i. In (19), we define SAMS as the set of candidate providers for each microservice of the application. An application has a maximum of t microservices.

$$SMS_i = \{SSP_{ik} \mid k \in [1, q]\} \tag{18}$$

$$SAMS = \{SMS_1, \ldots, SMS_t\} \tag{19}$$

4.2.3. Third Level

To host each microservice application, we must select the SAMS cloud providers (the result of the second level). For this, we must build candidate combinations for SAMS, so that each has a candidate combination for each microservice, and a set of these combinations is named SAPP (20). Moreover, in (20), $comb_{ik_ij_{k_i}}$ indicates the candidate combination j from the candidate provider k to the microservice i. Subsequently, (21) defines $SAPP_l.c$ as the execution cost for each combination in SAPP and checks whether it is less than or equal to the application execution cost, and excludes it otherwise. Next, we calculate each item score in SAPP. A candidate combination score ($SAPP_l.score$) is the score sum for each combination of items, as shown in (22).

$$\begin{aligned} SAPP = \{&(comb_{1k_1j_{k_1}}, \ldots, comb_{tk_tj_{k_t}}) \\ \mid &comb_{ik_ij_{k_i}} \in SSP_{ik}, SSP_{ik} \in SMS_i, SMS_i \in SAMS, \\ &1 \leqslant i \leqslant t, 1 \leqslant k_i \leqslant q_i, 1 \leqslant j_{k_i} \leqslant w_{k_i}\} \end{aligned} \tag{20}$$

$$SAPP_l.c = \sum_{i=1}^{t} comb_{ik_ij_{k_i}}.c + \sum_{i=1}^{t-1} Link_{k_i}.c \tag{21}$$

$$SAPP_l.score = \sum_{i=1}^{t} comb_{ik_ij_{k_i}}.score + \sum_{i=1}^{t-1} Link_{k_i}.score \tag{22}$$

Summarily, the selection process first selects the services of all providers that meet all requirements, which results in a set of candidate services from all available cloud providers. Then, we assemble all combinations of candidate services from all providers

that can be used to compose that microservice. Finally, we select one of the application combinations containing a provider to host each microservice based on a score. Therefore, as in our previous works [15–17], the selection process described for the LM^2K model can be considered a combinatorial optimization problem. Hence, we mapped the selection process to the multiple-choice knapsack problem as in our previous works [15–17].

5. Approaches for the LM^2K Model

In this section, we present and describe our proposed approaches for the LM^2K model. We mapped the selection process to the multi-choice knapsack problem and employed three algorithms suggested in the literature for this type of problem: dynamic (Section 5.1), greedy (Section 5.2), and ant colony (Section 5.3). We chose the dynamic algorithm to show an optimal solution, the greedy algorithm to accept larger data inputs, and the ant colony algorithm to obtain a better solution to the greedy (and a better data input than the dynamic). In this way, we can offer alternatives for the selection process according to the software architect's needs.

5.1. Dynamic Approach

In this subsection, we present the example and the algorithm for the dynamic approach of the LM^2K model.

5.1.1. Dynamic Approach Example

We present an example in Figure 5 that has three microservices and three candidate combinations for each one. The matrix presents the score, cost, and provider for each candidate combination. The software architect set a budget threshold of 12 for this application. Therefore, the matrix has 13 possible budgets (columns), ranging from 0 to 12. The filling of the matrix takes into account the communication links between the providers. We fill the last 13 columns by considering each candidate combination's cost and score, as well as the potential application budgets.

Microservice	Score	Cost	Provider	0	1	2	3	4	5	6	7	8	9	10	11	12
				0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0.6	2	P1	0	0	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	0.7	3	P2	0	0	0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.8	4	P3	0	0	0	0	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
2	0.5	2	P5	0	0	0.6	0.7	1.1	0.8	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	0.7	4	P4	0	0	0.6	0.7	0.8	0.8	1.3	1.4	1.5	0.8	1.5	1.5	1.5
	0.8	7	P3	0	0	0.6	0.7	0.8	0.8	0.8	0.8	0.8	1.4	1.5	1.6	1.6
3	0.6	1	P2	0	0.6	0.6	1.2	1.3	1.7	1.4	1.9	2.0	2.1	2.1	2.1	2.2
	0.7	3	P5	0	0	0.6	0.7	1.1	1.3	1.4	1.8	1.5	2.0	2.1	2.2	2.1
	0.8	8	P1	0	0	0.6	0.7	1.1	0.8	1.3	1.4	1.5	1.5	1.5	1.6	1.9

Figure 5. Dynamic approach example for LM^2K. The replaced scores are highlighted in pink. In blue are the scores that were affected by the altered scores. In orange is the selected combination for each microservice.

In Figure 5, we fill the first column with zeros, as well as the first row, which is an extra row that does not include any combination. To fill in each cell for the first microservice combination, we check the combination's cost. If it is higher than the budget value, we want to fill in the cell, we fill the cell with the previous cell value in the same column. Otherwise, we insert the combination score in the cell. The next step is to fill the cells corresponding to the other microservices. If the combination cost is higher than the budget value we want to fill, the cell value will be the highest value among the candidate combinations of the previous microservice from the same column.

In this example, we consider that the values of the requirements for providers P4 and P5 do not meet the application needs since we need to consider communication links. Thus,

we notice that combination 1 for microservice 2 belongs to provider P5 and 2 to provider P4. In column 5 for combination 1, the value should be 1.2 (0.5 + 0.7), the combination score plus the highest score of microservice 1 in column 3 (5 − 2). Therefore, this cell receives the highest score from microservice 1, in column 5. The same applies to combination 2 for microservice 2 in column 9 (in pink). The values of these cells influence the combination values 3 and 2, from columns 5 and 12, respectively, for microservice 3 (in blue).

Finally, we must select a combination for each microservice. For this, we check the combination for the last microservice (microservice 3) and select the highest score, which is column 12 combination 1. Then, we update the budget by reducing the cost of microservice 3 from the budget (12 − 1). For microservice 2, we repeat the first step and search for the highest score on the new budget, which is combination 3 in column 11. Since the combination for microservice 2 is 7, we now have a budget of 4 (11 − 7). Lastly, for microservice 3, the highest score comes from combination 3, in column 4. The final result is highlighted in orange. Furthermore, we notice the influence of the communication link between the providers.

5.1.2. Dynamic Algorithm

In this subsection, we describe the dynamic algorithm, divided into two parts as follows: the first part, illustrated by Algorithm 1, represents the first and second levels for the dynamic, greedy, and ant colony approaches; and the second part, which describes the third level and is illustrated by Algorithm 2.

Algorithm 1: Base Algorithm—Part 1.

```
input :Set of the application requirements ap
       Set of the available providers' capabilities cp
       Set of the communication links between services and providers lSer e lPr
       Execution flow set between the microservice tasks and microservices fSer e fPr
output:Set of providers prvdsMs to host microservices
1  sams ← initialize with empty set ; // list of candidate combinations with its score and its cost for all
     microservices
2  foreach ms of the ap do
3      ssp ← initialize with empty set ; // list of microservice candidate combinations in all providers
4      foreach sp of the cp do
5          candServ ← discoveryCandServ (ms,sp);
6          if (notempty(candServ)) then
7              matReq ← sawScalingPh (candServ);
8              candSerSC ← sawScorePh (candServ,matReq,ap.weight);
9              ssp ← ssp + (sp.number,combMsSp (candSerSC))
10         end
11     end
12     sms ← initialize with empty set ; // list of candidate combinations, score, and cost of a
         microservice in all providers
13     foreach comb of the ssp do
14         combSC ← calculateCombSC (comb,lSer,fSer);
15         combCost ← calculateCombCost (comb,lSer,fSer);
16         sms ← sms + (comb,combSC,combCost);
17     end
18     sams ← sams + (ms.number,sms);
19 end
```

Algorithm 1 takes a five-set input: (i) a set of application requirements, (ii) a set of capabilities of the available providers, (iii) a set of communication links from the available providers, (iv) the execution flow set for the microservice functions, and (v) the set of application microservices. The algorithm searches for the available provider services that meet all microservice requirements on line 5. Then, it sorts them using SAW on lines 7 and 8. Next, on line 9, it combines the services so that each combination has all of the services that a microservice requires. This process is repeated for all available providers. Furthermore, Algorithm 1 calculates the score and cost for each combination on lines 14 and 15, respectively. The algorithm calculates each combination's score and cost values, considering the microservice task execution flow, availability, response time, and cost value of each communication link between the services required for a microservice in the same provider. This process is repeated for all application microservices between lines 2 and

19. In the end, Algorithm 1 outputs a set of candidate combinations for each microservice (SAMS), each with a score and a cost.

Algorithm 2: Dynamic algorithm—Part 2.

```
1  sol ← initialize the matrix with 0;
2  last ← 0;
3  foreach sms of the sams do
4      for j ← (last + 1) to sizeof(sms) + last do
5          for cost ← 1 to ap.c do
6              if (combCost[j] > cost) then sol (cost,j) ← maxLine (sol (cost),ms [j-1].number);
7              else
8                  if ms[j].number == 1 then sol (cost,j) ← combSC[j];
9                  else
10                     if (ms [j ].number ≠ ms [j- 1].number) then
11                         (combSC,pr) ← discoveryPrComb (maxLine (sol (cost- combCost [j ]),ms
                             [j-1].number));
12                         res ← clAvaRT (combSC,pr,combSC [j ],ms [j ].number,lPr,fPr);
13                         if res == true then sol (cost,j) ← max ( combSC + combSC [j ], maxLine (sol
                             (cost),ms [j-1].number));
14                         else sol (cost,j) ← maxLine (sol (cost),ms [j-1].number);
15                     end
16                     else
17                         (combSC,pr) ← discoveryPrComb (maxLine (sol (cost- combCost [j ]),ms [j ].number
                             - 1));
18                         res ← clAvaRT (combSC,pr,combSC [j ],ms [j ].number,lPr,fPr);
19                         if res == true then
20                             sol (cost,j) ← max ( combSC + combSC [j ], maxLine (sol (cost),ms [j
                                 ].number-1));
21                         end
22                         else
23                             sol (cost,j) ← maxLine (sol (cost),ms [j ].number - 1);
24                         end
25                     end
26                 end
27             end
28         end
29     end
30     last ← sizeof(sms) + last;
31 end
```

Algorithm 2 outputs a set of providers to host each application's microservice. It represents the mapping of the selection process to the multi-choice knapsack problem. From lines 1 to 33, the algorithm selects a combination to host a microservice among the candidate combinations in SAMS (output of Algorithm 1). The algorithm creates a matrix (Sol) where each row contains a candidate combination for a microservice, and the columns represent the application budgets. The last column displays the budget threshold value defined by the software architect. The Sol Matrix is filled with the score values according to the candidate and the microservice it belongs to, ensuring that the combinations with the highest score, when combined, do not exceed the application's budget (AP.c). Thus, Algorithm 2 selects a combination to host a microservice among the candidate combinations in SAMS between lines 1 and 33. This process is illustrated in the example in Section 5.1.1.

5.2. Greedy Approach

In this subsection, we describe the greedy approach to our proposed model. We present an example and the algorithm that contains the details of the selection process.

5.2.1. Greedy Approach Example

Figure 6a presents the set of all candidate combinations for all application microservices, which has 4 candidate combinations for each of the 3 microservices.

Microservice	Combination	Score	Cost	Provider
1	1	0.4	1	P1
	2	0.3	3	P2
	3	0.9	4	P3
	4	1.0	6	P4
2	1	0.4	3	P5
	2	0.7	4	P4
	3	0.6	7	P2
	4	1.2	8	P3
3	1	0.3	1	P2
	2	0.7	3	P5
	3	0.9	7	P1
	4	0.6	8	P4

(**a**) Candidate Combinations.

Microservice	Combination	Score	Cost	Provider
1	1	0.4	1	P1
	2	**0.3**	**3**	**P2**
	3	0.9	4	P3
	4	1.0	6	P4
2	1	0.4	3	P5
	2	0.7	4	P4
	3	**0.6**	**7**	**P2**
	4	1.2	8	P3
3	1	0.3	1	P2
	2	0.7	3	P5
	3	0.9	7	P1
	4	**0.6**	**8**	**P4**

(**b**) Exclusion.

Microservice	Combination	Score	Cost	Provider
1	1	0.4	1	P1
	3	0.9	4	P3
	4	1.0	6	P4
2	1	0.4	3	P5
	2	0.7	4	P4
	4	1.2	8	P3
3	1	0.3	1	P2
	2	0.7	3	P5
	3	0.9	7	P1

(**c**) Selection.

Figure 6. Greedy approach example for LM^2K—part 1. The combinations that should be excluded are highlighted in the figure.

Figure 6a shows the score, cost, and provider of each candidate combination. The combination cost is used to order candidate combinations for each microservice. It shows the exclusion of candidate combinations from each microservice that is dominated or lp-dominated described in [36]. In this example, combinations 2, 3, and 4 of microservices 1, 2, and 3, respectively (highlighted in the figure), have higher costs than the previous combination, even though they have a lower score. Therefore, these combinations should be excluded. Figure 6c shows the candidate combinations that are suitable for selection.

The next step in the selection process is to calculate the incremental efficiency for all candidate combinations, except for the combinations with the lowest cost in each microservice, as shown in Figure 7a. Incremental efficiency is the cost–benefit of a candidate combination, and we use (23) to calculate it. In (23), `i` and `j` represent candidate combinations for a microservice, and `j` must be immediately after `i`. Moreover, *sc* and *cost* are the scores and the costs of the combinations. Then, the candidate combinations must be put in decreasing order, observing the incremental efficiency, except for the candidate combinations with lower costs for each microservice (in Figure 7b).

$$e = \frac{sc_j - sc_i}{cost_j - costi} \mid 1 < j \leqslant num(comb), i \leqslant j \tag{23}$$

Microservice	Combination	Score	Cost	Provider	Incremental Efficiency
1	1	0.4	1	P1	
	3	0.9	4	P3	0.4/3
	4	1.0	6	P4	0.3/2
2	1	0.4	3	P5	
	2	0.7	4	P4	0.3/1
	4	1.2	8	P3	0.5/4
3	1	0.3	1	P2	
	2	0.7	3	P5	0.4/2
	3	0.9	7	P1	0.2/4

(**a**) Incremental Efficiency.

Microservice	Combination	Score	Cost	Provider	Incremental Efficiency
2	2	0.7	4	P4	0.3/1
3	2	0.7	3	P5	0.4/2
1	3	0.9	4	P3	0.4/3
1	4	1.0	6	P4	0.3/2
2	4	1.2	8	P3	0.5/4
3	3	0.9	7	P1	0.2/4

(**b**) Sorting.

Figure 7. Greedy approach example for LM^2K—part 2.

The lowest cost combinations for each microservice compose the initial solution, and we subtract their respective costs from the application budget. In the example, combination 1 of each microservice makes up the initial solution, and the application budget threshold is 12, with the lowest combination costs being 1, 3, and 1, respectively. We then check the possibility of replacing each solution combination with one from the set ordered by the incremental efficiency in Figure 7b.

To perform the replacement, we must observe the requirement values of the communication links between providers. Thus, in Figure 8a, we substitute combination 1 for 2, both for microservices 2 and 3. However, when combination 1 is replaced by 2 in microservice 1, the communication link values do not meet the application's needs. Then, we must remove combination 2 for microservice 1. Next, we must continue the replacement process, but it is

not possible, as the next value (line 5 in Figure 7b) exceeds the budget. Thus, combination 1 is selected for microservice 1, as shown in Figure 8b.

Microservice	Combination	Score	Cost	Provider	Incremental Efficiency
1	1	0.4	1	P1	
	3	**0.9**	**4**	**P3**	**0.4/3**
	4	1.0	6	P4	0.3/2
2	1	0.4	3	P5	
	2	**0.7**	**4**	**P4**	**0.3/1**
	4	1.2	8	P3	0.5/4
3	1	0.3	1	P2	
	2	**0.7**	**3**	**P5**	**0.4/2**
	3	0.9	7	P1	0.2/4

(**a**) Communication Link Exclusion.

Microservice	Combination	Score	Cost	Provider	Incremental Efficiency
1	**1**	**0.4**	**1**	**P1**	
	4	1.0	6	P4	0.3/2
2	1	0.4	3	P5	
	2	**0.7**	**4**	**P4**	**0.3/1**
	4	1.2	8	P3	0.5/4
3	1	0.3	1	P2	
	2	**0.7**	**3**	**P5**	**0.4/2**
	3	0.9	7	P1	0.2/4

(**b**) Combination Selection.

Figure 8. Greedy approach example for LM^2K—part 3. The selected combinations for Microservices 2 and 3 are highlighted in blue and green, respectively. In orange is the removed combination for Microservice 1, and in yellow is the selected combination.

5.2.2. Greedy Algorithm

Algorithm 1 represents the first and second levels for the three approaches of the LM^2K model. Algorithm 3 illustrates the third level of the selection process using a greedy algorithm for the LM^2K model.

Algorithm 3: Greedy Algorithm—Part 2.

```
foreach sms of the sams do
    sorByCost (sms);
end
foreach sms of the sams do
    for j ← 0 to sizeof(sms) − 1 do
        if dominated (sms[j + 1], sms[j]) then remove (sms [j +1]);
    end
end
foreach sms of the sams do
    for j ← 0 to sizeof(sms) − 1 do
        if lp-dominated (sms[j + 2], sms[j + 1], sms[j]) then remove (sms [j +1]);
    end
end
foreach sms of the sams do
    budGet ← budGet − sms[0]cost ;
end
r ← [] ; i ← 0 ; k ← 0 ;
foreach sms of the sams do
    for j ← 1 to sizeof(sms) − 1 do
        SC ← sms[j].SC − sms[j − 1].SC ;
        Cost ← sms[j]cost − sms[j − 1]cost ;
        e ← sms[j] + (sms[j].SC)/(sms[j]cost) ;
        r[k] ← r[k] + (i, j, SC, Cost, e);
        k ← k + 1 ;
    end
    i ← i + 1 ;
end
sortDecreasingByE (r);
z ← [] ; i ← 0 ;
foreach sms of the sams do
    z ← z + (i, sms[0]sp, 1, sms[0].SC) ;
    i ← i + 1 ;
end
k ← 0 ;
repeat
    budGet ← budGet − r[k]cost ;
    foreach itemZ of the z do
        if (r[k][0] == itemZ[0]) then
            aux ← z;
            removeItem(itemZ, aux);
            insertItem(r[k], aux);
            res ← checkReqs(aux);
            if res == true then itemZ[3] ← r[k].SC;
        end
    end
    k ← k + 1 ;
until (budGet == 0 or budGet ≤ r[k]cost);
return (z) ;
```

In Algorithm 3, we classify candidate combinations for each microservice using the combination cost, line 2. From lines 4 to 8 and 9 to 13, we exclude each microservice's dominated and lp-dominated candidate combinations according to [36]. Between lines 14 and 16, we subtract the first candidate combination cost from the budget for each microservice. From lines 17 to 27, we created an instance of the knapsack problem. For this, on line 20, we assign to `SC` the difference between the combination score `j` and the combination score `j−1` for each microservice. On line 21, we assign to `cost` the difference between the combination cost `j` and the combination cost `j−1` for each microservice. On line 22, we calculate the incremental efficiency; on line 23, we add to `r` a tuple containing the microservice identifier `i`, the candidate combination identifier `j`, the score resulting from line 20, the cost resulting from line 21, and the incremental efficiency `e`. Then, on line 28, we sort `r` in decreasing order according to the incremental efficiency.

After identifying the candidate combinations with the highest profit, we build the final solution by placing the selected combinations in `Z`. Thus, from lines 29 to 33, we add a tuple to Z, which contains the microservice identifier `i`, the provider identifier `sms[0].sp`, the combination identifier (in this case, it is equal to 1), and the score of the first candidate combination of a microservice. Then, we obtain the combinations to host each microservice, between lines 34 and 41. In line 36, we update the budget by subtracting the cost `r[k]` from it. Between lines 37 and 39, we compare the microservice identifier of `r[k]` with the microservice identifier of `z`. If the communication link requirements meet the application needs, we update the item score `z` with the item score `r[k]`. Finally, we obtain `z`, which contains all service combinations to host all microservices, and each combination can belong to a different provider. This process can be seen in the example of Section 5.2.1.

5.3. Ant Colony Approach

According to [37], the ant colony algorithm is a multi-agent system in which there is a low level of interactions among agents, known as ants, resulting in complex behavior similar to that of a real ant colony. The algorithm is based on the concept of modeling the problem as a search problem, where artificial ants search for the path with the lowest cost. Pheromone, a distributed information type, is modified by the ants to reflect their accumulated experience during problem-solving. The amount of pheromone in a given path influences the choices of the ants: the higher the pheromone level, the more likely an ant is to select that path. This indirect means of communication aims to provide information about the quality of the path components to attract other ants. In the following sections, we will present an example and provide details of the selection process.

5.3.1. Ant Colony Approach Example

For the LM^2K model, each microservice represents a group of items, and each candidate combination ($Comb_{ij}$) is an item `i` of group `j`. Figure 9 shows the set of all candidate combinations for all application microservices, which has i_j candidate combinations for each microservice `j`. For each candidate combination of Figure 9, we add the score, the cost, the provider to which it belongs, and the initial pheromone. We must order the candidate combinations for each microservice by the combination cost. The process has `n` ants, and each of them must build an individual solution. In each solution, we consider the response time, availability, and cost for the communication links between the providers of each candidate combination selected by an ant. We repeat the process (`m` cycles), and each ant builds a solution in each cycle, generating `m` solutions. The highest profit solution in each cycle is selected, and the best solution is selected by the end of all cycles.

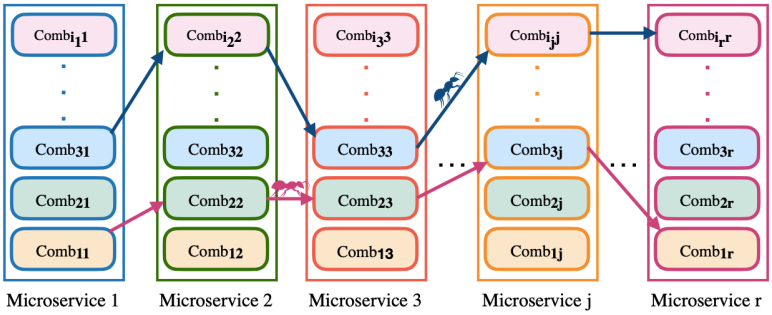

Figure 9. Example of the ant colony approach of model LM^2K.

5.3.2. Ant Colony Algorithm

The algorithm for the ant colony approach has two parts, similar to the other approaches for the model. The first contains the first and second levels, and Algorithm 1 describes them. This subsection describes Algorithm 4, which represents the third level of the ant colony approach. The budget threshold is the knapsack capacity, and the pheromone is the profit.

Algorithm 4: Ant Colony Algorithm—Part 2.

```
1  solution sApp ← initialize with empty set links lApp ← initialize with empty set repeat
2      solution sAnts ← initialize with empty set;
3      links lAnts ← initialize with empty set;
4      foreach eachAnt of the nAnts do
5          solution sTerm ← initialize with empty set;
6          links lTerm ← initialize with empty set;
7          foreach eachTerm of the flowTerm do
8              solution sMS ← initialize with empty set;
9              links lMS ← initialize with empty set;
10             foreach eachMS of the eachTerm do
11                 combCandidates ← Set of Candidate Combinations for eachMS;
12                 if eachMS == first microservice then To select a combination randomly from comb belongs to
                     combCandidates;
13                 else To select a combination from comb belongs to combCandidates with probability p;
14                 updatePheLocal (τ_comb,ρ,τ0);
15                 lMS ← { lMS ∪ prvdComb };
16                 sMS ← { sMS ∪ comb };
17             end
18             lTerm ← { lTerm ∪ lMS };
19             sTerm ← { sTerm ∪ sMS };
20         end
21         if (profit (sTerm,lTerm) > profit (sAnts,lAnts)) then
22             sAnts ← sTerm;
23             lAnts ← lTerm
24         end
25     end
26     if profit (sAnts,lAnts) > profit (sApp,lApp)) then
27         sApp ← sAnts;
28         lApp ← lAnts
29     end
30     updatePheGlobal (τ,ρ,sApp);
31 until m;
32 return (sApp)
```

The input for this algorithm consists of the application execution flow set, a set of communication link capabilities between providers, and a set of all candidate combinations for each microservice obtained from the first and second levels. Furthermore, the algorithm initializes a set of pheromones for each combination, representing the initial pheromone level, denoted as τ_0. In Equation (24), we assign τ_0 as the sum of the product between the normalized requirement value and its respective priority. The normalized requirement value is obtained by dividing the requirement value by the range between the minimum and maximum values. Thus, AMin, RTMin, and CMin are the minimum values for each requirement (availability, response time, and cost), while AMax, RTMax, and CMax are the maximum values. Furthermore, priA, priRT, and priC are the priorities for the availability, response time, and cost, respectively. The process has `m` iterations and `nAnts` ants. In each cycle of this algorithm, the `nAnts` ants build individual solutions.

The microservices belonging to the application execution flow are the knapsack groups. Thus, the algorithm selects the first microservice candidate from the execution flow. The algorithm chooses the candidate combination from the first group randomly. Then, it updates the local pheromone, according to τ_{ji}, as shown in (25), which is the local pheromone of candidate combination `j` from microservice `i`. In (25), we calculate τ_{ji} based on τ_0 and the growth parameter called (ρ). The value of (ρ) must be between $0 < \rho < 1$. For the other groups from the execution flow, the algorithm selects the most likely candidate combinations, according to (26).

An ant finishes building a solution when the solution contains a combination of each microservice from the execution flow. After each ant has created a solution, the algorithm identifies the best one for that iteration. Then, it updates the pheromone according to the best solution and according to τ, as shown in (27). We calculated τ based on ρ and the combination score. When the algorithm performs the maximum number of cycles, it ends.

The attractiveness of a combination depends on the pheromone deposited in it, and the combination selected in a group is the one that has the greatest attractiveness. The attractiveness of a combination is calculated by (26) [29]. In (26), β is a parameter that determines the relative importance of heuristic information and the pheromone level; the greater the β, the greater the influence of heuristic information in the selection process; q_0 is a parameter that determines the probability of choosing the best component and $0 < q_0 < 1$; q is a random number in the range $[0, 1]$; η_{ji} is the score of the selected combination. According to [38], if $q_0 = 1$, ants will choose only the component with the highest value since $q_0)$ will always be greater than q, which makes the algorithm a greedy search. However, if $q_0 = 0$, the algorithm will be more stochastic and less convergent.

$$\tau_0 = \frac{AMin}{AMax} \times priA + \frac{RTMin}{RTMax} \times priRT + \frac{CMin}{CMax} \times priC \tag{24}$$

$$\tau_{ji} = (1 - \rho) \times \tau_{ji} + \rho \times \tau_0 \tag{25}$$

$$p = \begin{cases} max_{ji}\{\eta_{ji}^{\beta} \times \tau_{ji}\} & \text{if } q \leqslant q_0 \\ \frac{\eta_{ji}^{\beta} \times \tau_{ji}}{\sum_{ji}\{\eta_{ji}^{\beta} \times \tau_{ji}\}} & \text{otherwise} \end{cases} \tag{26}$$

$$\tau = (1 - \rho) \times \tau + \rho \times score \tag{27}$$

6. Evaluation and Outcomes

In Sections 4 and 5, we described the model and approaches for selecting multi-cloud providers to host a microservice-based application based on the software architect's perspective. A single provider must host an application's microservice that best meets the microservice requirements. To evaluate the performance of the proposed approaches for the LM^2K model, in this section, we describe the tools implemented, the scenarios, and the experiments. Furthermore, we analyze the results obtained from the experiments.

6.1. Tool Description

We developed a tool for each proposed approach in the model. The tools were implemented using Python 3.7, and the input and output data were in the JavaScript Object Notation (JSON) format. For the LM^2K model, the tool requires three input files. The first file is divided into two parts: one containing the requirements of the microservice-based application, and the other containing the capabilities of the available cloud providers. The second file includes the execution flow among the application's microservices and the execution flow among the tasks within each microservice. The third file contains information about the capabilities of the interactions between the available cloud providers. The tool outputs a JSON file that contains the selected providers and services for hosting each microservice of the application.

6.2. Setting Scenarios

In this subsection, we describe the scenarios, experiments, and performance evaluation results for all approaches of the proposed model.

We conducted each experiment 30 times to ensure a normal distribution, as recommended by [39]. The scenarios were synthetically configured, aligning with common practices in the field, given the limited availability of datasets in the cloud computing research domain [40]. The providers in all scenarios were organized into sets based on the number of providers or provider services. We described twelve applications based on microservices, varying in the number of microservices, as presented in Table 2.

We assume that all microservices have the same size and require the same number of cloud services, but their requirements may vary. Each microservice requires three cloud services, which fall into different classes: computer, storage, and database. Consequently, the total number of cloud services required by an application is three times the number of microservices. Table 2 provides information on the number of microservices and the total services required for each configured application.

Most experiments use only five applications out of the twelve configured ones. Only experiments that check the influence of the number of microservices on the selection time of cloud services use most applications. Moreover, we set up three requirements for each microservice. The configured requirements are budget, response time, and availability, and we also determined the priority of each of them. We used them in all experiments and approaches. We performed the experiments on a MacBook Air with an i5 processor, 4 GB of RAM, and a 256 GB SSD.

Table 2. Applications for all experiments.

Applications	Characteristics	
	Number of Microservices	Number of Services
APP1	5	15
APP2	6	18
APP3	10	30
APP4	14	42
APP5	15	45
APP6	18	54
APP7	20	60
APP8	22	66
APP9	25	75
APP10	30	90
APP11	35	105
APP12	40	120

The application budget is the maximum value available to hire all cloud services needed for an application. The sum of all cloud service costs from all cloud providers used by the application is the application cost. In this manner, we also defined the budget classes, each with a different application budget cost, but the service price remained the same for all applications. Furthermore, we determined the service prices based on the minimum and maximum costs of cloud services described for the providers' capabilities.

We conducted experiments to explore variations in service costs by manipulating the budget classes. We defined availability and response time requirements within the capabilities of most providers. For instance, if we set the service cost to 10 and each microservice requires 3 services, the total cost for a single microservice would be 30. In a scenario where the application consists of 6 microservices, the budget would amount to 180.

Similarly, for an application with 9 microservices, the budget would be 270. Both budgets fall into the same budget class. Alongside the budget class, we also established requirement classes with varying values. These classes were configured to decrease the response time and cost as the availability requirement increased. Thus, a higher requirement class value indicates a greater degree of restriction.

We present all experiment results in graphs, in which the y-axis shows the average provider selection time in milliseconds (s), the lines represent each application, and the x-axis shows the experiment.

6.3. Dynamic Approach Experiments

This section evaluates the performance of the dynamic approach of the LM^2K model. To accomplish this, we established scenarios comprising the following components: (i) sets of providers (Table 3); (ii) capabilities of communication links among available providers; (iii) applications (Table 2); and (iv) the execution flow for each microservice application and its corresponding tasks, as described in Section 6.1. Table 3 presents eight sets of providers, with four sets differing in the number of providers and four sets differing in the number of services per provider. Furthermore, we configured the information pertaining to communication among the available providers, including availability, delay, and cost.

Table 3. Set of providers for the dynamic approach.

Approach	Characteristics of Sets				
	Number of Sets	**Number of Providers by Sets**	**Number of Services by Class**	**Number of Services by Provider**	**Number of Services by Set**
Dynamic LM^2K	4	5	4	12	60
			5	15	75
			6	18	90
			7	21	105
	4	5	4	12	60
		6			72
		7			84
		8			96

We evaluated the performance of the dynamic approach of the LM^2K model through seven experiments. For this, we used 8 out of the 12 configured applications (Table 2) for the experiment that varied the number of microservices. The applications used in this experiment were APP1, APP3, APP5, APP7, APP9, APP10, APP11, and APP12. We also utilized five configured applications (APP2, APP3, APP4, APP6, and APP8) for other experiments. The availability, response time, and cost variation requirements depend on the type of assessment.

For each application, the software architect must define the budget threshold, availability, and response time in addition to their weight. Moreover, the architect must configure execution flows among the application's microservices and each microservice. Thus, it must define each sequence among microservices and determine which microservices belong to the sequence, order, and frequency. A software architect must also specify the same information for the task sequences for each microservice.

From the configured scenarios, we defined experiments to evaluate the performance of the selection process. First, we conducted experiments by varying the number of providers, the number of services per provider, and the number of application microservices. In these experiments, the cost, availability, and response time requirements remained unchanged, and we defined values for them that should be met by most of the available providers. Figure 10 presents the results obtained from these experiments. In Figure 10a, we illustrate the results obtained by varying the number of providers. For this, we used the last four

sets of providers from Table 3, which differ in the number of providers and are represented on the x-axis of Figure 10a. We can observe that the selection time increases significantly with an increase in the number of providers, but after a certain threshold, the growth rate decreases.

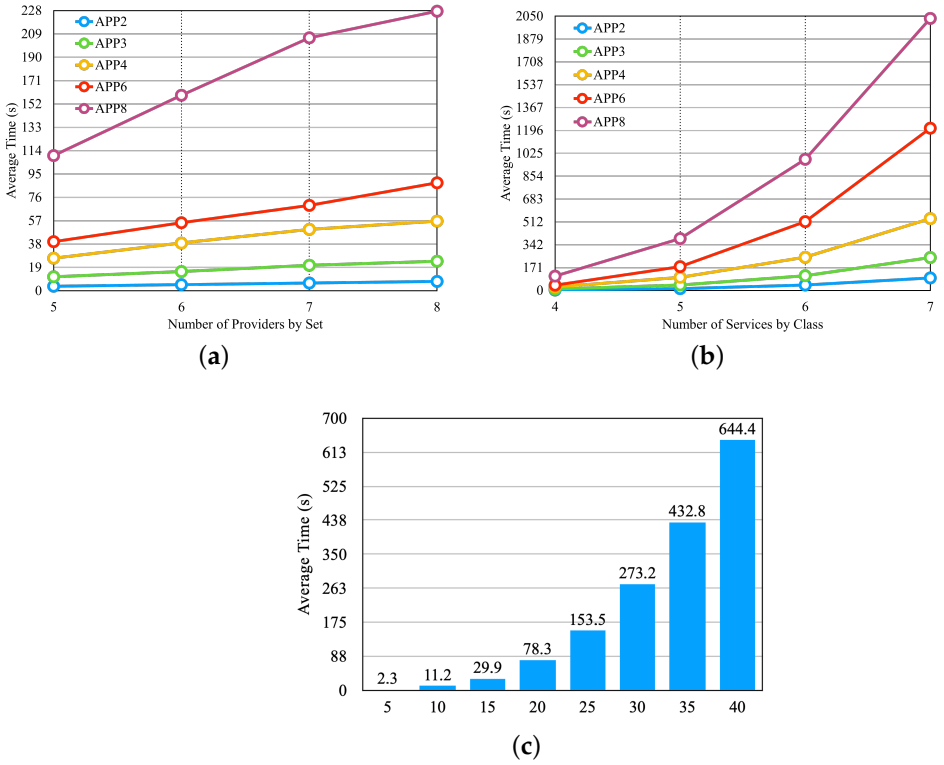

Figure 10. First three experiments of the dynamic approach—Table 3 sets. (**a**) Average Time per Set of Providers. (**b**) Average Time per Set of Provider Services. (**c**) Average Time per Microservices by Application.

Figure 10b shows the result of the experiment, in which we vary the number of services per provider. In this experiment, we use the first four providers in Table 3, which differ in the number of services per provider and are presented on the x-axis of Figure 10b. We can notice that the increase in the average selection time is large when increasing the number of services by providers. Figure 10c illustrates the result of the experiment, in which we vary the number of application microservices, and we used the set of providers of the fifth line in Table 3. In Figure 10c, the x-axis shows the number of microservices by application. We can observe that the number of microservices influences the outcomes.

From the results obtained in the first three experiments, we selected the set of the fifth row from Table 3 for another four experiments, consisting of five providers, with each provider offering four services per class. Figure 11 illustrates the results of these four experiments in separate graphs. In the experiment shown in Figure 11a, we configure the availability requirement with four different classes, where each class has the same percentage of variation in individual values based on the number of microservices in each application. The cost and response time requirements are set to values that should be met by most of the available providers. We observed that the availability values did not significantly influence the average selection time. This result can be attributed to the fact that in order to meet the defined availability value, cloud services need to have high availability values. As a result, the number of services with such high availability does not vary significantly between different availability values. Figure 11b examines the performance of the approach when varying the response time requirement, with the applications configured to have four different values. The availability and cost requirements

were set to values that should be met by most available providers. The results indicate that the response time values also do not significantly impact the average selection time, similar to the availability experiment.

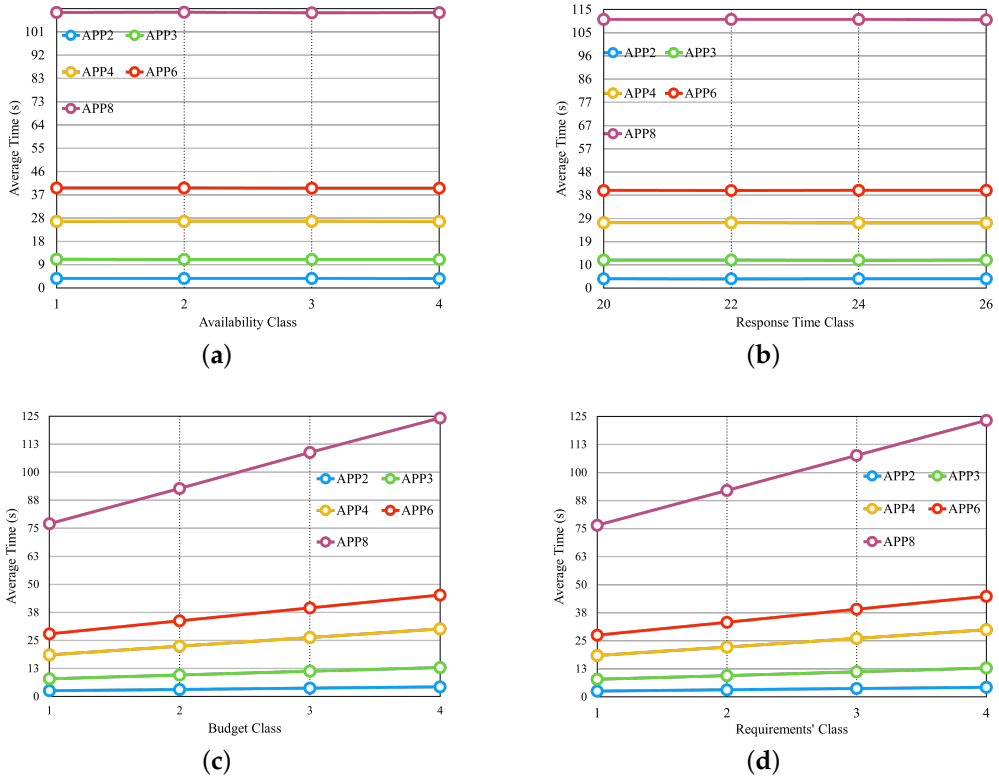

Figure 11. Four last experiments of the dynamic approach. (**a**) Average time for different classes of Availability. (**b**) Average time for different classes of Response Time. (**c**) Average time for different classes of the Budget Class. (**d**) Average time for different classes of the 'Requirements' Class.

In the experiment depicted in Figure 11c, we introduced budget classes as described in the preamble of Section 6.2. The results demonstrate that as the budget class increases, the average selection time also increases, which is in contrast to the findings of the previous two experiments. Therefore, the budget requirement has a significant impact on the average selection time. In the experiment shown in Figure 11d, where all three requirements are varied simultaneously, we created requirement classes as described earlier in Section 6.2. The results reveal that the average selection time increases as the requirements classes increase, primarily influenced by the cost requirement.

6.4. Greedy Approach Experiments

The purpose of this section is to verify the performance of the LM^2K model's greedy approach. To achieve this goal, we configured eight sets of providers, as shown in Table 4. Thus, four sets of providers differ by the number of providers, and four sets vary by the number of services per provider. In the sets that differ by the number of providers, each one has 25 services per class. The information defined for communication between providers includes delay, cost, and availability. We used 8 out of the 12 configured applications (Table 2) for the experiment that varied the number of microservices. The applications used in this experiment were APP1, APP3, APP5, APP7, APP9, APP10, APP11, and APP12. We used five of the configured applications for other experiments, i.e., APP2, APP3, APP4, APP6, and APP8. The requirements for availability, response time, and cost variation depend on the type of assessment.

After setting up all of the scenarios, we performed three experiments (in Figure 12). The first experiment verified the performance of the approach by varying the number of

providers, the second by varying the number of services by providers, and the third by varying the number of application microservices.

Table 4. Set of providers for the greedy approach of LM^2K.

Approach	Characteristics of Sets				
	Number of Sets	Number of Providers by Sets	Number of Services by Class	Number of Services by Provider	Number of Services by Set
Greedy LM^2K	4	10	25	75	1500
		20			1500
		30			2250
		40			3000
	4	15	20	60	900
			25	75	1125
			30	90	1350
			35	105	1575

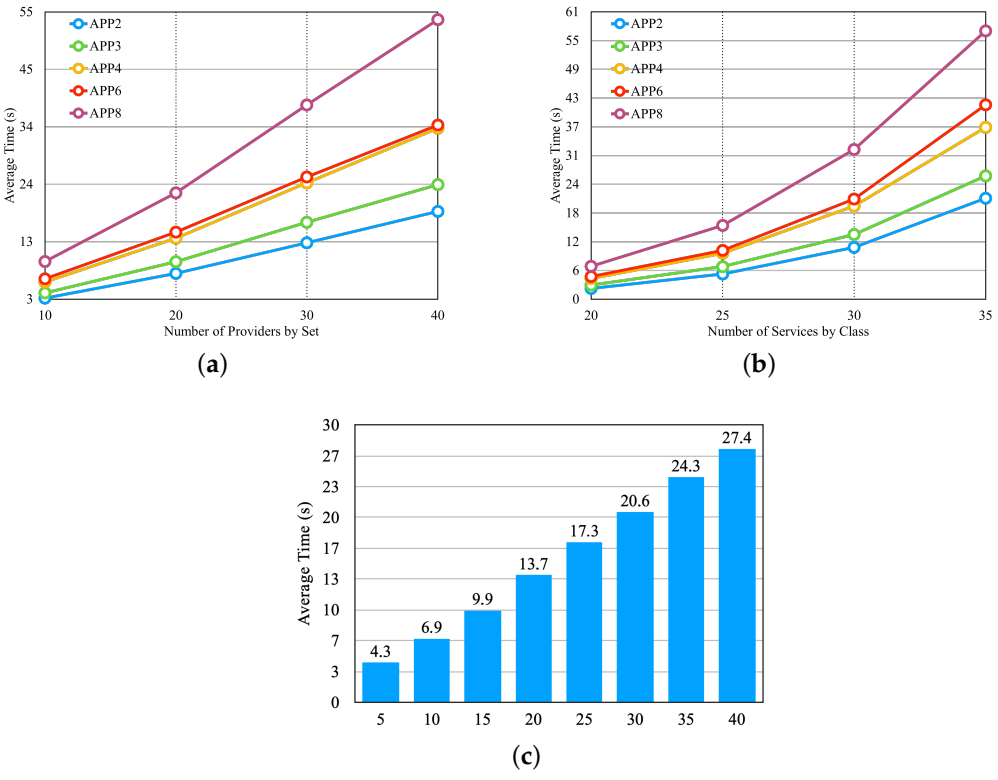

Figure 12. First three experiments for the greedy approach—Table 4 sets. (**a**) Average Time per Set of Providers. (**b**) Average Time per Set of Provider's Services. (**c**) Average Time per Microservices by Application.

In the graphs, the x-axis shows the values related to the experiments. In Figure 12a, the x-axis shows the number of providers according to the first four lines of Table 4. In Figure 12b, the x-axis shows the number of services according to the last four lines of Table 4. In Figure 12c, the x-axis shows the number of microservices by application and the set of providers of the sixth line in Table 4. In the graphs in Figure 12, we notice that the average selection time increases with the increase in the number of providers, the number of services per provider, and the number of application microservices, respectively.

From the first 3 experiments, we carried out 4 other experiments; we used a set of 15 providers, and each provider had 25 services per class, as described on the 5th line in Table 4. Figure 13 illustrates the experiments with a graph for each experiment.

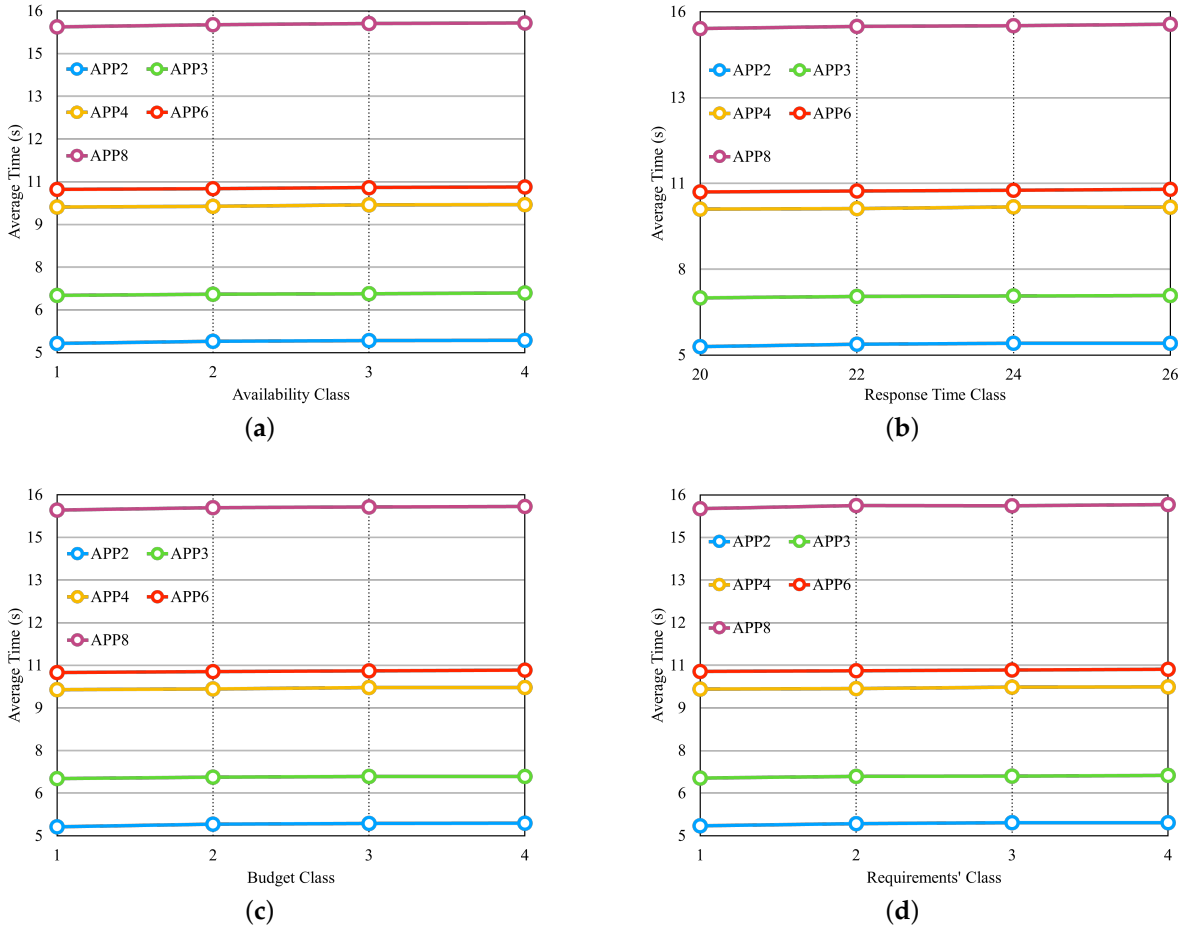

Figure 13. Four last experiments for the greedy approach. (**a**) Average time for different classes of Availability. (**b**) Average time for different classes of Response Time. (**c**) Average time for different classes of the Budget Class. (**d**) Average time for different classes of the 'Requirements' Class.

In the experiment displayed in Figure 13a, there are four different classes for the availability requirement, in which each class has the same percentage, but the values vary individually according to the number of microservices in each application. We defined the response time and cost requirements with fixed values that can meet most providers. In the experiment in Figure 13b, there are four different values for the response time requirement. Availability and cost requirements remain fixed with values that can meet most providers. Figure 13c presents the result of the experiment, and it is possible to notice that none of the applications had an influence on the selection time with the variation of the budget classes, which were described in the preamble of Section 6.2. Figure 13d shows the result of the experiment, with three requirements modified at the same time so that when the availability requirement increased, the requirements of the response time and cost decreased. Thus, a smaller class is less restricted, and a larger one is more restricted. In Figure 13, we can observe that none of the experiments influenced the average selection time. The values differed only by application, as each one had a different number of microservices.

6.5. Ant Colony Approach Experiments

This section investigates the performance of the ant colony-bioinspired approach of the LM^2K model. To achieve this, we define scenarios similar to the greedy approach of

this model. Table 5 shows four sets of providers, which differ by the number of providers, and another four sets that vary in the number of services per provider. The communication between providers is characterized by delay, cost, and availability. We configured the applications following the same structure as the greedy approach of the proposed models, as presented in Table 2. Additionally, we define the execution flows to be the same as those in the greedy approach of the LM^2K model, which were described in Section 6.4.

Table 5. Set of providers for the ant colony approach of LM^2K.

Approach	Characteristics of Sets				
	Number of Sets	Number of Providers by Sets	Number of Services by Class	Number of Services by Provider	Number of Services by Set
Ant Colony LM^2K	4	10	10	30	300
		20			600
		30			900
		40			1200
	4	15	10	30	450
			13	39	585
			15	45	675
			18	54	810

After setting up the scenarios, we carried out three experiments to verify the performance of the approach by varying the number of providers, the number of services per provider, and the number of application microservices. Figure 14 presents the results of the three experiments, respectively.

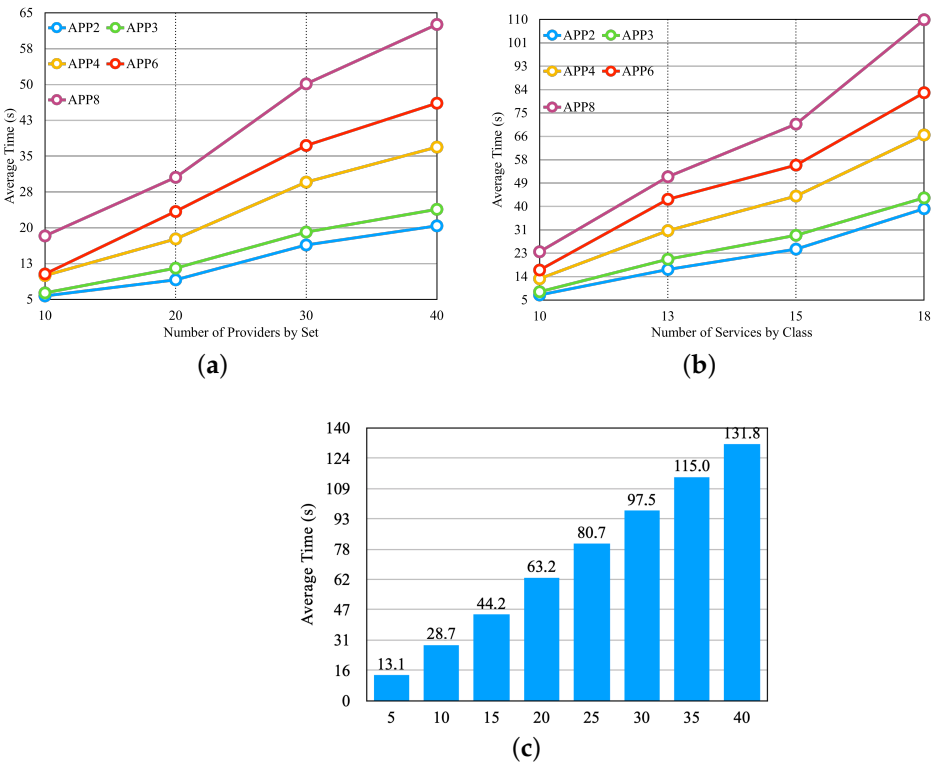

Figure 14. First three experiments for the ant colony approach—Table 5 sets. (**a**) Average Time per Set of Providers. (**b**) Average Time per Set of Provider's Services. (**c**) Average Time per Microservices by Application.

In the graphs, the x-axis shows the values related to the experiments. In Figure 14a, the x-axis shows the number of providers according to the first four lines of Table 5. In Figure 14b, the x-axis shows the number of services according to the last four lines of Table 5. In Figure 14c, the x-axis shows the number of microservices by application and uses the set of providers from the sixth line in Table 5. In the graphs of Figure 12, we notice that the average selection time increases with the increase in the number of providers, the number of services per provider, and the number of application microservices, respectively. We also observe that the number of microservices per application has a greater influence on the increase in the average selection time.

We performed another 4 experiments, for which, we used a set with 15 providers, and each provider had 25 services per class as described in the fifth line in Table 5. Figure 15 shows a graph for each experiment. All experiments have the same configuration as the experiments performed for the greedy approach of the LM^2K model presented in Section 6.4, except for the sets of providers. As in the experiments for the greedy approach, we notice that none of the experiments influenced the average selection time. The values differed only by application, as each of them has a different number of microservices.

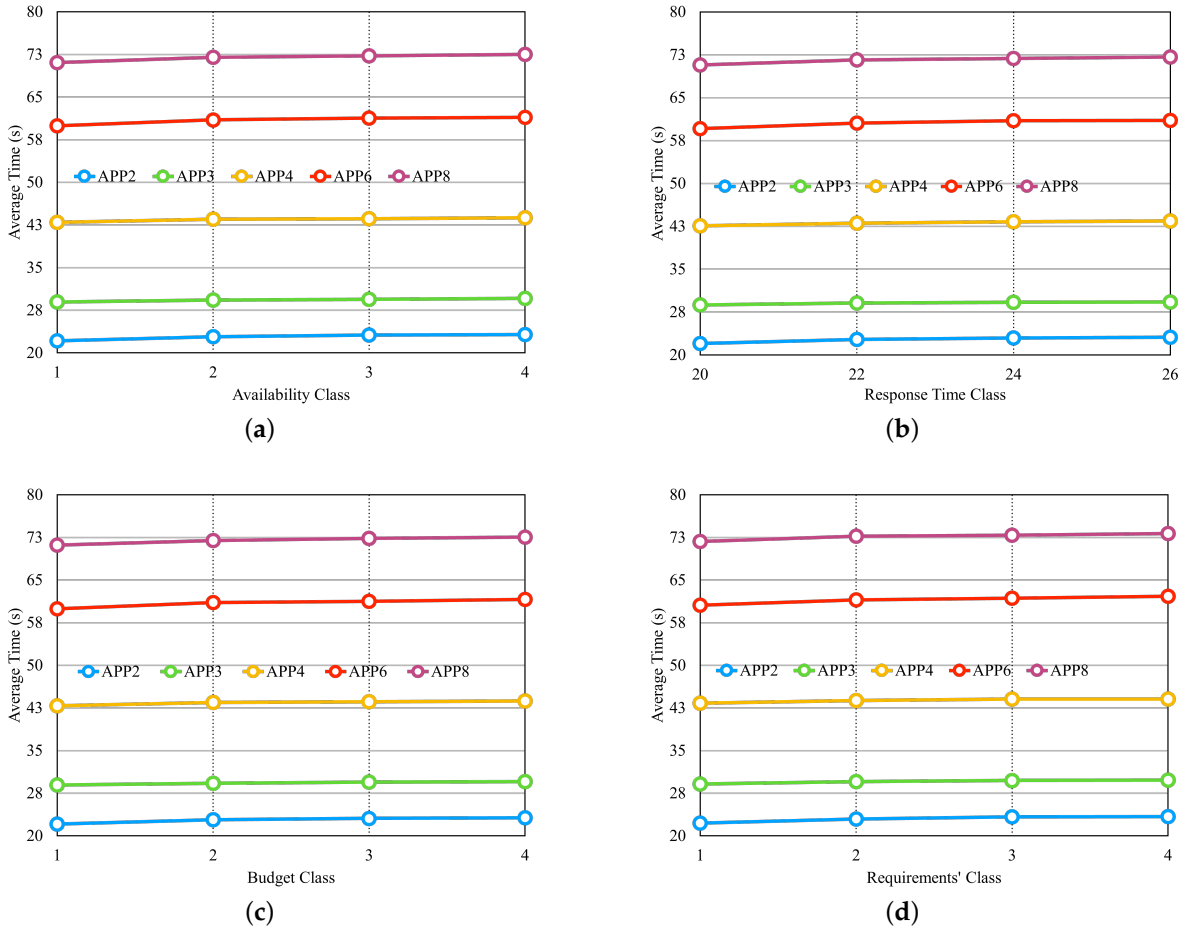

Figure 15. Four last experiments for the ant colony approach. (**a**) Average time for different classes of Availability. (**b**) Average time for different classes of Response Time. (**c**) Average time for different classes of the Budget Class. (**d**) Average time for different classes of the 'Requirements' Class.

6.6. Comparison of Outcomes

The LM^2K model addresses applications based on microservices that depend on each other and require communication among them. Consequently, the calculation of application requirements distributed in multiple clouds needs to consider the microservice execution flow, including the availability, delay, and cost of communication between providers.

Therefore, the approaches to the models need to address aspects that the other models did not consider. According to this context and the results obtained from the experiments, we observe that, in most outcomes, the average selection time does not change with the requirement variation. This result occurs because, in order to meet the thresholds defined by a software architect, providers must offer services with high values. Thus, the variation in the quantity of the offer has an insignificant amount in the average selection time. The exception is the dynamic approach for experiments that have budget variations, as they influence the average selection times and, thus, limit the size of the input for this approach. The greedy approach accepts a more considerable input than the ant colony approach, having an average time shorter than this. The ant colony approach allows for more significant input than the dynamic approach, and it also has shorter average selection times.

All experiments performed have varied at least one of the requirements or altered the number of providers per set or the number of services per provider. The experiments show the average time that each approach needs to select cloud providers to host the microservices of an application. We note that there is variation in the average selection time among applications because each application has a different number of microservices.

According to the experiments carried out, we also observed that there is a variation in the provider selection, the application requirement values, the number of providers, the number of services per provider, and even some changes in the provider's capabilities, which allows us to obtain better offers. This result shows the sensitivity of the approaches proposed. Therefore, a software architect must adequately define the application requirement values to obtain better offers.

7. Final Consideration and Future Works

The growth of cloud computing and, consequently, the increased number of providers and services have facilitated the software development process. Moreover, these cloud providers have specialized their services in a way that increases the chance of a user becoming a prisoner to that provider, i.e., not being able to migrate to another provider or distribute an application across multi-cloud providers. For a software architect to maximize the benefits of cloud computing, it is necessary to encourage the use of multi-cloud providers, with each microservice hosted by the provider that best meets the application's needs. Despite the many benefits of using multi-cloud providers, there are challenges, particularly in the absence of cooperation agreements among the involved providers in a multi-cloud environment.

One of the first decisions that a software architect must make for a traditional application is to choose the cloud providers to host each component, as the implementation depends on the provider selected. However, applications developed for the cloud must use an architectural style that brings greater flexibility, such as microservices. Thus, an application based on microservices facilitates its distribution to multi-cloud providers. Nevertheless, multi-cloud providers offer services with the same functionalities, but different capabilities, and each application's microservice may need multiple cloud services. It is up to the software architect to perform the arduous task of selecting a service combination for each microservice from all available cloud providers.

In this work, we proposed a new model called Link Microservice Mapped to the Knapsack (LM^2K) and presented three approaches for selecting providers to host application microservices from the software architect's perspective. These approaches include a dynamic algorithm, a greedy algorithm, and an ant colony algorithm.

The results showed that all proposed approaches are viable. However, each approach demands a different number of providers and application microservices. The ant colony's algorithm cannot deal with input sizes as big as the greedy, though it usually provides better results than the greedy solution. Furthermore, the dynamic approach will provide the optimal solution but can only work with scenarios with a small number of providers, services, and microservices. Therefore, each approach has a particular suitability that

depends on the number of providers, services, and microservices. Designing a specific scenario to compare them would not be a reasonable comparison.

In this work, we analyzed the approaches proposed for selecting multiple providers. Future work will involve conducting a more detailed analysis of the services and providers chosen in each approach. This analysis will examine how the selection may change based on variations in requirement values or priorities. Additionally, we plan to compare the proposed approaches based on the quality of the services selected in each experiment. The quality of services refers to the values and priorities of the requirements.

Author Contributions: Investigation , J.C., D.V. and F.T.; Writing—original draft, J.C.; Writing—review & editing, C.R. and D.V.; Supervision, D.V. and F.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Linked Microservice Mapped to the Knapsack Equation Nomenclature

Table A1. Description of equations' nomenclature.

Description	Equation	Nomenclature
Availability (.*a*)	(1)	$AP.a$
Availability (.*a*) of *j*-th application term (.*a*)	(2)	$Term_j.a$
Availability (.*a*) of sequence *i* of the term *j*	(3)	$Seq_{ij}.a$
Maximum Response Time Threshold (.*rt*)	(4)	$MaxRT$
Response Time (.*rt*) of Application	(5)	$AP.rt$
Response Time (.*rt*) of *j*-th term	(6)	$Term_j.rt$
Response Time (.*rt*) of *i*-th sequence of the *j*-th term	(7)	$Seq_{ij}.rt$
Application Execution **Cost** (.*c*)	(8)	$AP.c$
Cost (.*c*) of *j*-th term	(9)	$Term_j.c$
Cost (.*c*) of *i*-th sequence of the *j*-th term	(10)	$Seq_{ij}.c$
Requirements Score of each candidate service	(13)	$Score(S_i)$
Set of candidate combinations from the provider	(14)	SSP_{ik}
Combination (*comb*) of services for the *i*-th microservice	(15)	$comb_{ikj}$
Score (.*score*) of services' combinations	(16)	$comb_{ikj}.score$
Cost (.*cost*) of services' combinations	(17)	$comb_{ikj}.c$
Candidate providers' set for the *i*-th microservice	(18)	SMS_i
Candidate combinations from *k*-th provider to *i*-th microservice	(18)	SSP_i
Set of all candidate combinations of providers for application microservices	(19)	$SAMS$
Candidate combination for each microservice	(20)	$SAPP$
Execution **Cost** (.*c*) for each $SAPP$ combination	(21)	$SAPP_l.c$
Score (.*score*) for each $SAPP$ combination	(22)	$SAPP_l.score$
Incremental **Efficiency** (*e*): cost–benefit of a candidate combination	(23)	e

References

1. Vaquero, L.M.; Rodero-Merino, L.; Caceres, J.; Lindner, M. A break in the clouds. *ACM Sigcomm Comput. Commun. Rev.* **2009**, *39*, 50. https://doi.org/10.1145/1496091.1496100.
2. Opara-Martins, J.; Sahandi, R.; Tian, F. Critical review of vendor lock-in and its impact on adoption of cloud computing. In Proceedings of the International Conference on Information Society, i-Society 2014, London, UK, 10–12 November 2014; pp. 92–97. https://doi.org/10.1109/i-Society.2014.7009018.
3. Mezgár, I.; Rauschecker, U. The challenge of networked enterprises for cloud computing interoperability. *Comput. Ind.* **2014**, *65*, 657–674. https://doi.org/10.1016/j.compind.2014.01.017.
4. Sousa, G.; Rudametkin, W.; Duchien, L. Automated Setup of Multi-Cloud Environments for Microservices-Based Applications. In Proceedings of the 9th IEEE International Conference on Cloud Computing, Francisco, CA, USA, 27 June–2 July 2016. https://doi.org/10.1109/CLOUD.2016.49.

5. Wang, Y.; He, Q.; Ye, D.; Yang, Y. Service Selection Based on Correlated QoS Requirements. In Proceedings of the IEEE International Conference on Services Computing (SCC), Honolulu, HI, USA, 25–30 June 2017; pp. 241–248. https://doi.org/10.1109/SCC.2017.38.
6. Jatoth, C.; Gangadharan, G.R.; Buyya, R. Optimal Fitness Aware Cloud Service Composition using an Adaptive Genotypes Evolution based Genetic Algorithm. *Future Gener. Comput. Syst.* **2019**, *94*, 185–198. https://doi.org/10.1016/j.future.2018.11.022.
7. Petcu, D. Multi-Cloud: Expectations and Current Approaches. In Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds, Prague, Czech Republic, 22 April 2013; ACM: New York, NY, USA, 2013; pp. 1–6. https://doi.org/10.1145/2462326.2462328.
8. Petcu, D. Consuming Resources and Services from Multiple Clouds: From Terminology to Cloudware Support. *J. Grid Comput.* **2014**, *12*, 321–345. https://doi.org/10.1007/s10723-013-9290-3.
9. Toosi, A.N.; Calheiros, R.N.; Buyya, R. Interconnected Cloud Computing Environments. *Acm Comput. Surv.* **2014**, *47*, 1–47. https://doi.org/10.1145/2593512.
10. Grozev, N.; Buyya, R. Inter-Cloud architectures and application brokering: taxonomy and survey. *Softw. Pract. Exp.* **2014**, *44*, 369–390. https://doi.org/10.1002/spe.2168.
11. Petcu, D. Portability in Clouds: Approaches and Research Opportunities. *Scalable Comput. Pract. Exp.* **2014**. *15*, 251–270. https://doi.org/10.12694/s...ARTICLE.
12. Carvalho, J.O.D.; Trinta, F.; Vieira, D. PacificClouds: A Flexible MicroServices based Architecture for Interoperability in Multi-Cloud Environments. In Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018), Funchal, Portugal, 19–21 March 2018; pp. 448–455. https://doi.org/978-989-758-295-0.
13. Sethi, M. *Cloud Native Python: Build and Deploy Applications on the Cloud Using Microservices, AWS, Azure and More*; Packt Publishing Ltd.: Birmingham, UK, 2017.
14. Ziade, T. *Python Microservices Development*; Packt Publishing Ltd.: Birmingham, UK, 2017.
15. Carvalho, J.; Vieira, D.; Trinta, F. Dynamic Selecting Approach for Multi-cloud Providers. In *Cloud Computing—CLOUD 2018, Proceedings of the 11th International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, 25–30 June 2018*; Luo, M., Zhang, L.J., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 37–51.
16. Carvalho, J.; Vieira, D.; Trinta, F. Greedy Multi-cloud Selection Approach to Deploy an Application Based on Microservices. In Proceedings of the 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Pavia, Italy, 13–15 February 2019. https://doi.org/10.1109/PDP.2019.00021.
17. Carvalho, J.; Vieira, D.; Trinta, F. UM2Q: multi-cloud selection model based on multi-criteria to deploy a distributed microservice-based application. In Proceedings of the 10th International Conference on Cloud Computing and Services Science, Online, 7–9 May 2020; SCITEPRESS-Science and Technology Publications: Setubal, Portugal, 2020; pp. 56–68.
18. Sosinsky, B. *Cloud Computing Bible*; Wiley Publishing Inc.: Hoboken, NJ, USA, 2011; p. 473.
19. NIST. *NIST Cloud Computing Standards Roadmap*; Technical Report; NIST: Gaithersburg, MD, USA, 2011.
20. Gavvala, S.K.; Jatoth, C.; Gangadharan, G.R.; Buyya, R. QoS-aware cloud service composition using eagle strategy. *Future Gener. Comput. Syst.* **2019**, *90*, 273–290. https://doi.org/10.1016/j.future.2018.07.062.
21. Sun, L.; Dong, H.; Hussain, O.K.; Hussain, F.K.; Liu, A.X. A framework of cloud service selection with criteria interactions. *Future Gener. Comput. Syst.* **2019**, *94*, 749–764. https://doi.org/10.1016/j.future.2018.12.005.
22. Khanam, R.; Kumar, R.R.; Kumar, C. QoS based cloud service composition with optimal set of services using PSO. In Proceedings of the 4th IEEE International Conference on Recent Advances in Information Technology, RAIT 2018, Dhanbad, India, 15–17 March 2018; pp. 1–6. https://doi.org/10.1109/RAIT.2018.8389039.
23. Kumar, S.; Bahsoon, R.; Chen, T.; Li, K.; Buyya, R. Multi-Tenant Cloud Service Composition Using Evolutionary Optimization. In Proceedings of the International Conference on Parallel and Distributed Systems—ICPADS, Singapore, 11–13 December 2018; pp. 972–979. https://doi.org/10.1109/PADSW.2018.8644640.
24. Ding, S.; Wang, Z.; Wu, D.; Olson, D.L. Utilizing customer satisfaction in ranking prediction for personalized cloud service selection. *Decis. Support Syst.* **2017**, *93*, 1–10. https://doi.org/10.1016/j.dss.2016.09.001.
25. Jatoth, C.; Gangadharan, G.; Fiore, U.; Computing, R.B. SELCLOUD: A hybrid multi-criteria decision-making model for selection of cloud services. *Soft Comput.* **2018**, *23*, 4701–4715. https://doi.org/https://doi.org/10.1007/s00500-018-3120-2.
26. Panda, S.K.; Pande, S.K.; Das, S. SLA-based task scheduling algorithms for heterogeneousmulti-cloud environment. *Arab. J. Sci. Eng.* **2018**, *43*, 913–933. https://doi.org/10.1007/s13369-017-2798-2.
27. Moghaddam, M.; Davis, J.G. Simultaneous service selection for multiple composite service requests: A combinatorial auction approach. *Decis. Support Syst.* **2019**, *120*, 81–94. https://doi.org/10.1016/j.dss.2019.03.005.
28. Liu, L.; Gu, S.; Zhang, M.; Fu, D. A hybrid evolutionary algorithm for inter-cloud service composition. In Proceedings of the 9th International Conference On Modelling, Identification and Control, ICMIC 2017, Kunming, China, 10–12 July 2017; pp. 482–487. https://doi.org/10.1109/ICMIC.2017.8321692.
29. Liu, H.; Xu, D.; Miao, H.K. Ant colony optimization based service flow scheduling with various QoS requirements in cloud computing. In Proceedings of the 1st ACIS International Symposium on Software and Network Engineering, SSNE 2011, Seoul, Republic of Korea, 19–20 December 2011; pp. 53–58. https://doi.org/10.1109/SSNE.2011.18.
30. Zhou, J.; Yao, X.; Lin, Y.; Chan, F.T.; Li, Y. An adaptive multi-population differential artificial bee colony algorithm for many-objective service composition in cloud manufacturing. *Inf. Sci.* **2018**, *456*. https://doi.org/10.1016/j.ins.2018.05.009.

31. Hongzhen, X.; Limin, L.; Dehua, X.; Yanqin, L. Evolution of Service Composition Based on Qos under the Cloud Computing Environment. In Proceedings of 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS), Chongqing, China, 28–29 May 2016; pp. 66–69.
32. Zeng, L.; Benatallah, B.; Ngu, A.H.; Dumas, M.; Kalagnanam, J.; Chang, H. QoS-aware middleware for Web services composition. *IEEE Trans. Softw. Eng.* **2004**, *30*, 311–327. https://doi.org/10.1109/TSE.2004.11.
33. Alrifai, M.; Risse, T. Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition. In Proceedings of the 18th International Conference on World Wide Web, Madrid, Spain, 20–24 April 2009; ACM: New York, NY, USA, 2009; pp. 881–890. https://doi.org/10.1145/1526709.1526828.
34. Canfora, G.; Di Penta, M.; Esposito, R.; Villani, M.L. An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, Washington, DC, USA, 25–29 June 2005; ACM: New York, NY, USA, 2005. https://doi.org/10.1145/1068009.1068189.
35. Seghir, F.; Khababa, A. A hybrid approach using genetic and fruit fly optimization algorithms for QoS-aware cloud service composition. *J. Intell. Manuf.* **2016**, *29*, 1773–1792.. https://doi.org/10.1007/s10845-016-1215-0.
36. Kellerer, H.; Pferschy, U.; Pisinger, D., The Multiple-Choice Knapsack Problem. In *Knapsack Problems*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 317–347. https://doi.org/10.1007/978-3-540-24777-7_11.
37. Goldbarg, M.C.; Goldbarg, E.; Luna, H.P.L. *Otimização Combinatória e Meta-Heurísticas*; Elsevier: Rio de Janeiro, Brazil, 2016.
38. Wei-Neng Chen.; Jun Zhang. An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements. *IEEE Trans. Syst. Man Cybern. Part* **2009**, *39*, 29–43. https://doi.org/10.1109/TSMCC.2008.2001722.
39. Hogendijk, J.; Whiteside, A.E.S.D. *Sources and Studies in the History of Mathematics and Physical Sciences*; Springer: Berlin/Heidelberg, Germany, 2011; p. 415. https://doi.org/10.1007/978-0-387-87857-7.
40. Hayyolalam, V.; Pourhaji Kazem, A.A. A systematic literature review on QoS-aware service composition and selection in cloud environment. *J. Netw. Comput. Appl.* **2018**, *110*, 52–74. https://doi.org/10.1016/j.jnca.2018.03.003.