UM^2Q: Multi-cloud Selection Model based on Multi-criteria to Deploy a Distributed Microservice-based Application

Juliana Carvalho[1], Dario Vieira[2] and Fernando Trinta[3]
[1]*Federal University of Piauí (UFPI), Picos, Brazil*
[2]*EFREI-Paris, Paris, France*
[3]*Federal University of Ceará (UFC), Fortaleza, Brazil*

Keywords: Multi-cloud Selection, Microservices, Multi-criteria Decision-making Method.

Abstract: Choosing the best configuration to deploy a distributed application in a multi-cloud environment is a complex task since many cloud providers offer several services with different capabilities. In this paper, we present a multi-cloud selection process to deploy a distributed microservice-based application, which has low communication cost among microservices. The proposed selection process is part of PacificClouds, an approach that intends to manage the deployment and execution of distributed applications across multiple providers from the software architect perspective. The proposed approach selects various providers, where each provider must host an entire microservice with multiple tasks consuming several cloud services. The UM^2Q approach selects the provider that better meets the software architect requirements, for each microservice of a multi-cloud application. Hence, the proposed process of selecting multiple providers uses multi-criteria decision-making methods to rank the cloud services and selects cloud providers and services by individually observing each microservice requirement, such as cloud availability, response time, and cost. Further, we propose a formal description of UM^2Q and a brief one of the strategy implementation. We also introduce a set of experiments to evaluate UM^2Q performance, and the outcomes showed its feasibility for a variable number of requirements, microservices and providers, even for extreme values.

1 INTRODUCTION

Nowadays, Cloud Computing has become a very popular model to provide IT resources. Many cloud companies are widely available to offer several resources such as virtual servers, storage, and entire applications to their users. Cloud Computing market is very competitive, and users have a variety of offerings to select the service that best fits their requirements. In contrast, cloud providers try to provide several kinds of services and resources, leading to a situation where they became specific features. This scenario brings a severe challenge for cloud applications known as vendor lock-in.

A new and more challenging scenario for cloud application developers is called multi-cloud, where a single application is composed of resources or services hosted by different cloud providers. Multi-cloud enables developers to build their systems with components that better fit their needs, but, it also brings new challenges. For instance, an application may require resources available on different cloud providers, where each resource has different costs, performance, or availability offers. Hence, multi-cloud brings more flexibility to software architects where they can select services for providers that better serve their system design. However, since these applications are more complex, using multiple clouds brings challenges for choosing the best set of resources among the available ones as there may be several providers offering many services with the same functionalities, but different capabilities. This scenario makes the process of selecting the best providers a hard task.

As aforementioned, to facilitate the software architects decision-making, Carvalho et al., 2018b proposed PacificClouds, which is a new approach to manage the deployment and execution process of an application based on microservices in multi-cloud environment from the software architect perspective. The multi-cloud environment aims to mitigate vendor lock-in and also allows the software architect to distribute an application to take the advantages offered by cloud computing. Also, PacificClouds uses microservices as the architectural style because they pro-

Carvalho, J., Vieira, D. and Trinta, F.
UM2Q: Multi-cloud Selection Model based on Multi-criteria to Deploy a Distributed Microservice-based Application.
DOI: 10.5220/0009338200560068
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 56-68
ISBN: 978-989-758-424-4

vide support to build applications in which each part is independently scalable and deployed. To achieve its goals, PacificClouds must select the providers and services to host all application microservices.

A cloud selection process must require several microservices to build an application, and each of them needs various cloud services to compose your activities. Besides, a cloud selection process for PacificClouds must select among available cloud providers, which one better meets the needs of the software architect and the microservice, observing the functionalities and requirements of each service required to compose each microservice. Further, PacificClouds must deploy each microservice in a distinct single cloud provider.

The unlinked microservice mapped to quota (UM^2Q) approach described in this work uses a multi-criteria decision-making (MCDM) method to rank all cloud services that meet all requirements in each provider. Also, UM^2Q selects providers observing each microservice needs regardless of the other microservices that compose an application. In this paper, we only address applications based on microservices, which have low communication cost among microservices. We described the entire the proposed selection process in Section 3. In addition, the approach proposed in this paper is part of the PacificCloud project (Carvalho et al., 2018b), which encompasses two other approaches to solve the problem of selecting multiple cloud providers (Carvalho et al., 2018a), (Carvalho et al., 2019). We compare them in Section 5.

While our work addresses the multi-cloud selection process to deploy microservices from the software architect's perspective, there are other works in the literature that also deal with the cloud selection problem, but they focus on other issues: some select services in a single cloud provider, e.g., Chen et al., 2016 and Hongzhen et al., 2016, while others deal with the problem using cloud federation, e.g., Thomas and Chandrasekaran, 2017 and Panda et al., 2018; some rank cloud services, e.g., Ding et al., 2017 and Jatoth et al., 2018, and others address the cloud selection to compose services, e.g., Bharath Bhushan and Pradeep Reddy, 2016 and Mezni and Sellami, 2017; some focus on a trust evaluate, e.g.,Tang et al., 2017 and Somu et al., 2018, and other works focus on cloud manufacturing, e.q., Zheng et al., 2016 and Zhou et al., 2018. In Section 5, we describe and compare some of these research works.

In pursuance of addressing the issues described above, we propose UM^2Q, a multi-cloud selection process which will be used by PacificClouds to deploy the application microservices. The main contributions of this work are as follows:

1. UM^2Q selects one provider for each application microservice from the software architect perspective, in which each provider selects all necessary cloud services to meet every microservice.
2. We propose a formal description of the selection process in Subsection 3.1, which describes how to select each cloud service for each microservice.
3. UM^2Q uses a multi-criteria decision-making (MCDM) method to rank the cloud services in each available provider and considers the software architect requirements and its priorities. We described the method in Subsection 3.2.
4. In Subsection 3.2, UM^2Q selects providers by individually observing each microservice, which avoids the need for a global combination of candidate services and consequently consumes less time during providers selection.
5. In Section 4, we performed an evaluation of the selection process performance and the outcome shows the feasibility of UM^2Q regardless of the application requirements, number of microservices, and number of providers.
6. We performed a comparison of UM^2Q with other works that address the cloud selection problem from the software architect perspective. For this, we proposed Providers and Services Granularity (PSG), which is a taxonomy to deal with issues regarding the number of providers in the selection process, the number of services in each provider, and the selection process result (Section 5).

2 MOTIVATION

As aforementioned, the primary goal of this work is to lead the multiple clouds selection process for PacificClouds. In this section, we show how PacificClouds should select multiple clouds for each microservice. Besides, as PacificClouds intends to manage the deployment and execution process of an application based on microservices in multi-cloud, we present the definitions of multiple clouds and microservices adopted in our work.

Figure 1 shows the deployment plan generation service (DPGS) to PacificClouds, which is responsible for selecting the cloud providers and services to host the microservices. We observe that the cloud providers selection has three steps: first, it receives all requirements from the SLA service; then, it receives the cloud providers capabilities from the Adapter; and finally, it selects a provider for each microservice among the candidate providers that better meets its requirements through multi-cloud selection service. For this, it observes a set candidate cloud services in each provider to optimize the software architect's requirements. Each application microservice is deployed in the candidate provider that better meets the requirements. Finally, it sends the selected providers to deployment plan generation. Therefore, the use of the multi-cloud is very important to allow each microservice to find the best offer according to its requirements, and to mitigate vendor lock-in. The selection approach has three levels. The first one selects the required cloud services for each microservice in each provider. The second one selects the candidate combinations to compose each microservice in each provider. The third one selects a provider among candidate providers for each microservice.

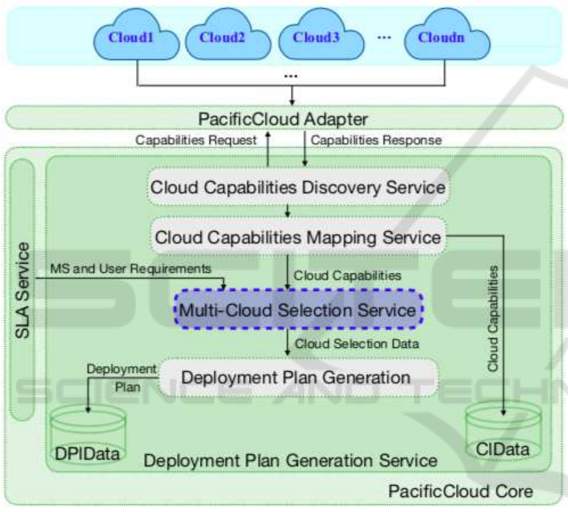


Figure 1: The multi-cloud Selection by PacificClouds.

We adopt the definition of the multi-cloud used by Petcu, 2014, who classifies different multiple cloud application scenarios as delivery models. One of them is multi-cloud, in which the services can be used in parallel or sequentially, and it does not include a prior agreement among the cloud providers and a third party acts as a mediator. We also adopt the definition of microservices used by Carvalho et al., 2018b, where a microservice is a set of autonomous, independent, self-contained services, in which each service has a single goal, is loosely coupled, and interacts to build a distributed application. Microservices represent an application business function.

3 THE SELECTION MODEL

According to Section 2, we can notice that there may be many available cloud providers, and each of them offers several services. Also, an application can have several microservices, and each one of them may require several cloud services to meet the microservices tasks needs. Therefore, the task of selecting cloud providers to host a distributed application in multiple clouds is complex.

In this section, we propose a multi-cloud selection model for PacificClouds to host an application based on microservices called UM^2Q. The proposed selection model refers to the cloud providers selection from the software architect's perspective before an application deployment. The proposed selection process is based on the software architect requirements. Thus, the software architects must define their requirements for an application. Also, the software architects must set a budget' quota for each application microservice. We consider that the provider's capabilities are available. Even though our approach can handle as many requirements as needed, in this paper, we consider three user requirements: (i) response time, (ii) availability, and (iii) application execution cost.

3.1 Formulation

As mentioned above, the required cloud services to compose a microservice have different requirements and the providers offer many services with the same functionalities but different capabilities. In this work, we choose to assess application response time (execution time + 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 understand the proposed selection process. However, the software architect can define several other user requirements without significant changes in the code.

According to the providers selection process previously described, we propose:

- Definition 1: **Cloud Service Model** - as S(S.rt,S.a,S.c), in which S.rt, S.a, and S.c stand for response time, availability, and cost, respectively, as in (Liu et al., 2011).
- Definition 2: **Cloud Services 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 same functionalities but different capabilities.
- Definition 3: **Services Provider Model** - as $SP_k = \{SC_{k1}, SC_{k2}, \ldots, SC_{kp}\}$, in which SC_{k1}, SC_{k2}, …, SC_{kp} are services classes.

- Definition 4: **Cloud Provider Set** - as CP = $\{SP_1, SP_2, \ldots, SP_q\}$, in which SP_1, SP_2, …, SP_q are services 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 indispensable to execute a microservice, and they should be of different cloud service classes. Each service required to compose the microservice `i` can belong to different classes of services from a cloud provider. Thus, $1 \leqslant k_r \leqslant o$, in which `o` is the maximum number of classes for a cloud provider.
- Definition 6: **Application Model** - as AP = $\{MS_1, MS_2, \ldots, MS_t\}$, in which $MS_1, MS_2, \ldots, MS_t$ are microservices.

The main goal of the multi-cloud selection process is to select the providers that satisfy software architects requirements and optimize their specified objectives, as follows:

3.1.1 Availability Requirement

The cloud availability for an application must meet at least the user-defined threshold, so that each application microservice meets the same threshold. We define `MinAvbty` as the user-defined minimum availability threshold. In Eq. 1, we define `AP.a` as the application availability, which is assigned the lowest availability value among its microservices, and it must be greater than or equal to `MinAvbty`. In addition, in Eq. 2, $MS_i.a$ represents the microservice availability `i`, which is assigned the lowest availability value among their services and it must be greater than or equal to `MinAvbty`. The cloud availability for a service is represented by $S_{ij_x}^{m_{kx}}.a$ in Eq. 2, which is the service `j` of the microservice `i`.

$$AP.a = \min_{1 \leqslant i \leqslant t} (MS_i.a) \geqslant MinAvbty, \forall MS_i \mid MS_i \in AP \tag{1}$$

$$\begin{gathered} MS_i.a = \min_{1 \leqslant j \leqslant r} (S_{ij}^{k_j}.a) \geqslant MinAvbty, \\ \forall S_{ij}^{k_j} \mid S_{ij}^{k_j} \in MS_i, MS_i \in AP, 1 \leqslant i \leqslant t \end{gathered} \tag{2}$$

3.1.2 Response Time Requirement

The response time for an application must meet the user-defined threshold, so that each application microservice meets the same threshold. We define `MaxRT` as the user-defined maximum response time threshold, and `MaxRT` is the maximum execution time threshold (`MaxExecTime`) plus the maximum delay threshold (`MaxDelay`) defined by the user, as in Eq. 3. In Eq. 4, we define `AP.rt` as the response time of the application, which is assigned the highest response time value among their microservices, and that must be less than or equal to `MaxRT`. In addition, in Eq. 5, $MS_i.rt$ represents the response time of microservice `i`, which is assigned the highest response time value among their services and that must not be less than `MaxRT`. The response time for a service is represented by $S_{ij}^{k_j}.rt$ in Eq. 5, which is the service `j` of the microservice `i`.

$$MaxRT = MaxExecTime + MaxDelay \tag{3}$$

$$AP.rt = \max_{1 \leqslant i \leqslant t} (MS_i.rt) \leqslant MaxRT, \forall MS_i \mid MS_i \in AP \tag{4}$$

$$\begin{gathered} MS_i.rt = \max_{1 \leqslant j \leqslant r} (S_{ij}^{k_j}.rt) \leqslant MaxRT, \\ \forall S_{ij}^{k_j} \mid S_{ij}^{k_j} \in MS_i, MS_i \in AP, 1 \leqslant i \leqslant t \end{gathered} \tag{5}$$

3.1.3 Cost Requirement

The application execution cost should not be higher than the cost threshold defined by the user (`Budget`). In Eq. 6, we define `AP.c` as the application execution cost, which is assigned as the sum of all its microservices's costs, and that must be less than or equal to the provided Budget.

$$AP.c = \sum_{i=1}^{t} MS_i.c \leqslant Budget, \forall MS_i \mid MS_i \in AP \tag{6}$$

In this work, an application has `t` as the microservices maximum, and a microservice has `r` as the services maximum. We do not address the services capabilities model from the provider perspective because this work focuses on the software architect perspective. We only verify if the service capabilities offered by a cloud provider meet the user requirement.

3.2 UM^2Q Selection Process

In UM^2Q, we select multi-cloud providers to host each microservice of an application and each microservice is hosted by a single provider, considering only the microservice and software architect requirements. Each microservice is hosted in a single provider for it to have a low cost of communication between services from different providers. This approach presents three selection levels, which will be described next.

3.2.1 First Level

We select the services of each provider that meet all software architect requirements which results in a candidate services set for each provider. Next, we rank all candidate services in each provider. We use Simple Additive Weighting (SAW) technique as in (Seghir and Khababa, 2016), 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, with 1 = availability, 2 = response time, 3 = cost. These candidate services refer to the same service of the microservice. We must perform the entire process for each service of the microservice. Each row R_i corresponds to a cloud service S_{ij} and each column R_j corresponds to a requirement. Next, the requirements should be ranked using one of the two criteria described in Eqs. 7 and 8. Also, $R_j^{Max} = Max(R_{ij}), R_j^{Min} = Min(R_{ij}), 1 \leqslant i \leqslant n$.
Negative: the higher the value, the lower the quality.

$$V_{ij} = \begin{cases} \frac{R_j^{Max} - R_{ij}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \quad (7)$$

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

$$V_{ij} = \begin{cases} \frac{R_{ij} - R_j^{Min}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \quad (8)$$

Weighting Phase. The overall requirements score is computed for each candidate cloud service (Eq. 9).

$$Score(S_i) = \sum_{j=1}^{3} (V_{ij} * W_j) \mid W_j \in [0,1], \sum_{j=1}^{3} W_j = 1 \quad (9)$$

At the end of the first level, we have a set of all candidate services for each microservice in all available providers.

3.2.2 Second Level

We must select all required services to compose each microservice in each provider from the candidate services selected in the first level. For this, we consider that the software architects define a budget quota for each microservice of an application. Thus, in Eq. 10, we define $MS_i.c$ as the microservice execution cost, which must be less than or equal to $(Budget * Quota_i)$. We define $Quota_i$ as the application execution budget quota defined for microservice `i`, and Budget as the application execution cost threshold. Next, in Eq. 11, we define that each $Quota_i$ must be between 0 and 1 and that the value sum of all $(Budget * Quota_i)$ must be equal to the Budget. In addition, in Eq. 12, we define that the cost sum of all services of microservice `i` must be less than or equal to the execution cost of microservice `i`.

$$MS_i.c \leqslant Budget * Quota_i, \forall MS_i \mid MS_i \in AP \quad (10)$$

$$Quota_i \in]0,1], \sum_{i=1}^{t} Budget * Quota_i = Budget \quad (11)$$

$$\sum_{j=1}^{r} S_{ij}^{k_j}.c \leqslant Budget * Quota_i, \forall S_{ij}^{k_j} \mid S_{ij}^{k_j} \in MS_i, 1 \leqslant i \leqslant t \quad (12)$$

In order to compose a microservice, we need to combine all candidate services and check the execution cost for these services combinations. First, we must combine the candidate services that come from the same microservice and are offered by the same provider, which is represented by $(S_{ij_1}^k, ..., S_{ij_r}^k)$ in Eq. 13. Each element of the candidate combination $S_{ij}^{k_j}$ represents the candidate service `j` of provider `k` to the microservice `i`, in which $1 \leqslant j \leqslant r$ and `r` indicates the number of services to microservice `i`. Next, we calculate the combination execution cost and verify if it is less than or equal to $(Budget * Quota_i)$ as shown in Eq. 13, which is in accordance to Eqs. 10, 11, and 12. Each microservice has a maximum `r` services.

$$(S_{i1}^{k_1}, ..., S_{ir}^{k_r}) \mid (S_{i1}^{k_1}.c + ... + S_{ir}^{k_r}.c) \leqslant Budget * Quota_i \quad (13)$$

Next, we must choose a combination among candidate services combinations in each provider for each microservice. For this, it calculates the average score for each services combination, which is represented by aSC_{SP_k} in Eq. 14. SP_k is the provider `k` that belongs to a set of providers (CP), and CP has a maximum `q` providers. Besides, $Score(S_{i1}^{k_1}) + Score(S_{i2}^{k_2}) + ... + Score(S_{ir}^{k_r})$ represents the sum of all service scores of a combination, $S_{ir}^{k_r}$ is a candidate service of a combination, and `r` is a maximum of microservice services. We must choose the combination with the highest score average.

$$aSC_{SP_k} = \frac{Score(S_{i1}^{k_1}) + Score(S_{i2}^{k_2}) + ... + Score(S_{ir}^{k_r})}{r} \quad (14)$$

If multiple combinations have the highest score average, one of them must be selected based on one of the three requirements according to the priorities defined by a software architect.

At the end of the second level, there must be a candidate provider set (CSP_i) for each MS_i of an application as shown in eq. 15. SP_{ki} represents the candidate provider `k` of microservice `i`, and it has a candidate service combination chosen for a microservice `i`. An application has a maximum `t` microservices, and a microservice has a maximum q_i candidate providers.

$$CSP_i = \{SP_{1i}, SP_{2i}, \ldots, SP_{q_i i}\} \mid 1 \leqslant i \leqslant t, \quad t = sizeof(AP), q_i = sizeof(CSP_i) \tag{15}$$

3.2.3 Third Level

We select one provider for each microservice. First, we check the microservice execution cost for each candidate provider, which was calculated in the second level, as shown in Eq. 16. In this equation, $SP_{ki}.c$ indicates the execution cost of the microservice `i` for candidate provider k. We define that $SP_{ki}.c$ is the service costs sum of the candidate combination of provider `k`. In Eq. 16, $S_{ij}^{kj}.c$ is the service `j` cost of microservice `i` in candidate provider `k`.

$$SP_{ki}.c = \sum_{j=1}^{r}(S_{ij}^{kj}.c) \mid S_{ij}^{kj} \in SP_{ki}, SP_{ki} \in CSP_i, \quad 1 \leqslant k \leqslant q_i, 1 \leqslant i \leqslant t \tag{16}$$

Next, we check the CSP from the Second Level for each microservice to obtain the SP that presents the average microservice execution cost, and we use Eqs. 17, 18 and 19. In Eq. 17, $Max(MS_i.c)$ returns the highest microservice execution cost among the candidate providers for microservice `i`. Next, in Eq. 18, $Min(MS_i.c)$ returns the lowest microservice execution cost among the candidate providers for microservice `i`. In addition, in Eq. 19, $Average(MS_i.c)$ returns the average microservice execution cost among the candidate providers for microservice `i`.

$$Max(MS_i.c) = \max_{1 \leqslant k \leqslant s_i}(SP_{ki}.c) \mid \forall SP_{ki} \in CSP_i, \quad 1 \leqslant i \leqslant t, q_i = sizeof(CSP_i) \tag{17}$$

$$Min(MS_i.c) = \min_{1 \leqslant k \leqslant s_i}(SP_{ki}.c) \mid \forall SP_{ki} \in CSP_i \quad 1 \leqslant i \leqslant t, q_i = sizeof(CSP_i) \tag{18}$$

$$Average(MS_i.c) = \frac{Max(MS_i.c) + Min(MS_i.c)}{2} \tag{19}$$

In case there is more than one provider with the same average execution cost, we select the provider that presents the highest availability or performance, observing the priority defined by a user. For this, we use Eqs. 20 and 21 to calculate the availability and response time for each candidate provider, respectively. Equation 20 defines $SP_{ki}.a$ as the cloud availability of provider `k` for microservice `i`, which is assigned the lowest candidate service availability for microservice `i`. Equation 21 defines $SP_{ki}.rt$ as the response time of provider `k` for microservice `i`, which is assigned the highest candidate service response time of microservice `i`. Next, we use Eqs. 22 and 23 to calculate the highest value for availability and the lowest value for response time in each candidate provider based on Eqs. 20 and 21, respectively.

$$SP_{ki}.a = \min_{1 \leqslant j \leqslant r}(S_{ij}^{kj}.a) \mid S_{ij}^{kj} \in SP_{ki}, SP_{ki} \in CSP_i, \quad 1 \leqslant k \leqslant q_i, 1 \leqslant i \leqslant t \tag{20}$$

$$SP_{ki}.rt = \max_{1 \leqslant j_x \leqslant r}(S_{ij_x}^{k}.rt) \mid S_{ij_x}^{k} \in SP_{ki}, SP_{ki} \in CSP_i, \quad 1 \leqslant k \leqslant q_i, 1 \leqslant i \leqslant t \tag{21}$$

$$Max(MS_i.a) = \max_{1 \leqslant k \leqslant s_i}(SP_k.a) \mid \forall SP_k \in CSP_i \tag{22}$$

$$Max(MS_i.rt) = \min_{1 \leqslant k \leqslant s_i}(SP_k.rt) \mid \forall SP_k \in CSP_i \tag{23}$$

3.3 Diagram and Algorithm

In this subsection, we present a diagram and describe an algorithm for UM^2Q, which are illustrated in Figure 2 and Algorithm 1, respectively. Figure 2 shows an overview of each level described in Subsection 3.2 to the multi-cloud selection process. Algorithm 1 presents the pseudocode with more detail of every level of our approach.

The diagram in Figure 2 starts with a set of available cloud providers and a set of application microservices, and it has three parts representing each level of our approach. It shows two tasks in this first part: the first one discovers every candidate service for a microservice in a cloud provider, and the second one ranks all candidate services.

The second part of the diagram shows four tasks, which aim to build all the candidate combinations for a microservice in all available cloud providers. First, it makes all candidate combinations in a single provider. A candidate combination is composed of all necessary services for a microservice, and the combination cost is lower than or equal to Budget's quota. Next, it calculates the combination score and selects the combination with the highest score. If there is more than one combination with the highest score, then it chooses the combination that has the highest priority. The entire process is repeated for all available providers.

The third diagram part in Figure 2 presents four tasks, which aim to select a cloud provider combination for each microservice. In the first and second tasks, it calculates the combination cost and selects the combination with the average cost. If there is more than one combination with the average cost, it selects the combination with the highest priority. The whole process is repeated for all microservices of an application. Finally, it returns a set of combinations, in which each combination represents a microservice.

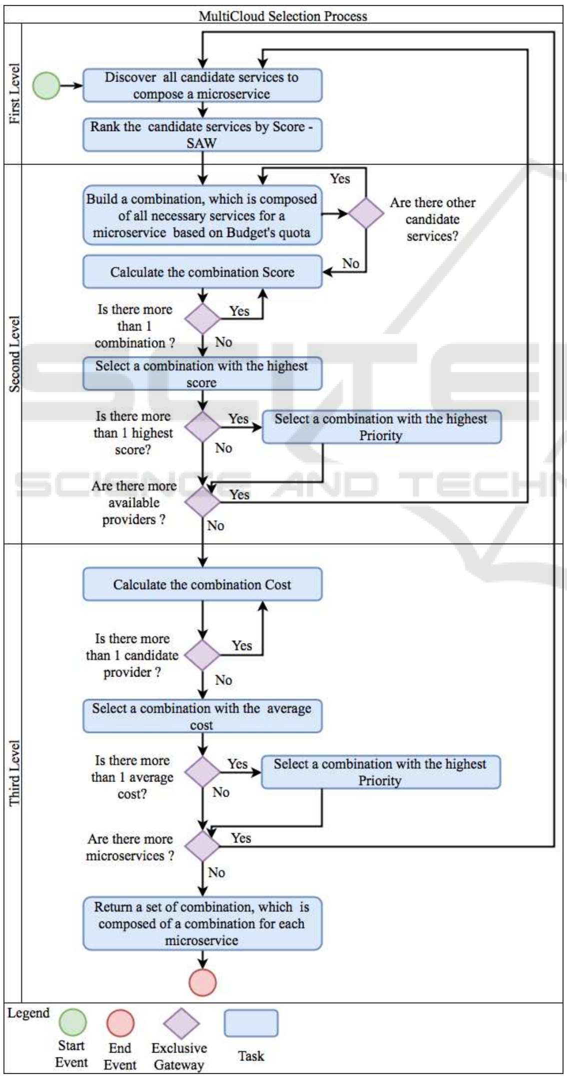

MultiCloud Selection Process
First Level
Discover all candidate services to compose a microservice
Rank the candidate services by Score - SAW
Second Level
Build a combination, which is composed of all necessary services for a microservice based on Budget's quota
Are there other candidate services?
Yes
No
Calculate the combination Score
Is there more than 1 combination ?
Yes
No
Select a combination with the highest score
Is there more than 1 highest score?
Yes
No
Select a combination with the highest Priority
Are there more available providers ?
Yes
No
Third Level
Calculate the combination Cost
Is there more than 1 candidate provider ?
Yes
No
Select a combination with the average cost
Is there more than 1 average cost?
Yes
No
Select a combination with the highest Priority
Are there more microservices ?
Yes
No
Return a set of combination, which is composed of a combination for each microservice
Legend
Start Event
End Event
Exclusive Gateway
Task

Figure 2: Diagram for UM^2Q.

Next, we describe Algorithm 1, which is based on Figure 2. It has two inputs: the set of application requirements and the set of cloud providers capabilities; and one output: the set of the providers to host all application microservices.

To achieve the goals of our approach, Algorithm 1, in line 6, discovers the candidate services in a single provider, in which a candidate service is a cloud service that meets all requirements for a microservice. Next, in lines 8 and 9, it ranks the candidate services, according to Eqs. 7, 8, and 9. After, in line 10, it combines all candidate services necessary to compose a microservice in a manner that the combination cost is lower than or equal to the microservice quota in accordance to Eqs. 10, 11, 12, and 16. Then, in line 11, it adds all candidate combinations of a provider to the candidate combinations list for a microservice. It calculates the combination score in accordance to Eq. 14 between lines 13 and 17. It returns the combination with the highest score in line 19, next, it checks if there is more than one combination with the highest score between lines 20 and 24. In case this happens, it checks which of these combinations has the highest priority, according to user preferences. The entire process is repeated in every available provider for the same microservice. Hence, at the end of this level, it has a set of combinations for a microservice, being one of each candidate cloud provider. A candidate cloud provider is a cloud provider that has at least one candidate services combination to compose a microservice.

Note that in line 29, to achieve the third level, Algorithm 1 calculates the average combination cost according to Eqs. 17, 18, and 19. Next, in line 30, it verifies the combinations that have the average cost. In lines 31 and 32, if there is more than one combination with the average cost, it checks which combination has the greatest score in line 32. If there is more than one combination with the highest score in line 34, it gets the combination with the highest priority according to user preferences. Thus, at the end of this level, it has a combination of a provider to host a microservice. The entire process is repeated for all microservices of an application. It returns a set of combination, which is composed of a combination for each microservice.

4 EVALUATION

In this work, we evaluate the UM^2Q process performance. First, we set up the scenarios that allow us to check UM^2Q behavior so that it is the same for the other scenarios. Next, we developed a tool to evaluate UM^2Q feasibility. In this section, we describe the scenarios configuration, the tool, the experiments, and the outcomes.

Algorithm 1: The Algorithm for UM^2Q.

```
input   : Set of the application requirements ap
          Set of the capabilities of the providers
          cp
output  : Set of Providers prvdsMs to host each
          application microservice
foreach ms of the ap do
    combLtPr ← initialize with empty set;
    combsMS ← initialize with empty set;
    combListMS ← initialize with empty set;
    foreach sp of the cp do
        candServ ← discoveryCandServ (ms, sp);
        if (not empty(candServ)) then
            matReq ← sawScalPh (candSr);
            candSrSC ← sawSCPh (candSr, matReq,
              ap.wgt);
            combsMS ← combsMS + combMsSp
              (candSrSC, quotaMS);
            combLtPr ← combLtPr + (sp.number,
              combsMS);
        end
        combLtPrSC ← initialize with empty set;
        foreach comb of the combLtPr do
            combSC ← calculateCombSC (comb);
            combLtPrSC ← combLtPrSC + (comb,
              combSC);
        end
        combLtPrHg ← initialize with empty set;
        combLtPrHg ← combLtPrHg + combHgtSC
          (combLtPrSC);
        if (not empty(combLtPrHg)) then
            if (sizeof(combLtPrHg) > 1) then
                combPrMS ← combHgtPrty
                  (combLtPrHg, prty);
            end
        end
        combLtMS ← combLtMS + combPrMS;
    end
    if (not empty(combLtMS)) then
        if (sizeof(combLtMS) > 1) then
            avgCost ← averageCost (combLtMS);
            combLtMSAgCost ← combAvgCost
              (combLtMS, avgCost);
            if (sizeof(combLtMSAgCost) > 1) then
                finalCbMS ← finalCbMS +
                  combHgtSC (combLtMS);
                if (sizeof(finalCbMS) > 1) then
                    finalCbMS ← combHgtPrty
                      (combLtMS, prty);
                end
            end
        end
    end
    combFinalLt ← combFinalLt + (ms.number,
      finalCbMS);
end
return combLtFinal
```

4.1 Tool Description

We developed a tool to evaluate the UM^2Q selection process. The tool was implemented using Python 3.7, and we used the JSON format as input and output. The tool has two JSON files as input: one contains the service providers capabilities; the other has the application requirements. Each provider capability is organized by service classes and each class has its services. We consider three classes: computing, storage, and database. Each service must contain its functional and non-functional capabilities. As described in Section 3, UM^2Q considers availability, response time, and cost as nonfunctional capabilities.

Application information involves name, minimum availability, maximum response time, maximum budget, weight and priority of each user requirement, which are the same for each microservice. Besides, an application also contains microservice information, and there must be a description of the name and the total budget quota as in Eq. 10. The tool returns a JSON file, which contains the providers that will host the microservices as well as the services that will be used and each providers' cost.

4.2 Setting Scenarios

According to Hayyolalam and Pourhaji Kazem, 2018, there are few available datasets in the research domain. In this manner, in fifty percent of the works referenced by Hayyolalam et al, the authros use a dataset configured randomly. We also randomly configured ten sets of providers, which differ from one another by the number of providers and service capabilities. Each provider has three services classes: compute, storage, and database; each class has seven services. The amount of providers in each set is a multiple value of 100, in which the first set has 100 providers and the last 1000.

Also, we configured the requirements for microservices-based applications. We configured five applications: the first one (APP1) has 6 microservices, the second (APP2) 9, the third (APP3) 12, the fourth (APP4) 15, and the last one (APP5) has 18 microservices. Each microservice contains one service of each class: compute, storage, and database. Availability, response time, and cost requirements vary depending on the type of assessment that should be performed.

4.3 Experiments and Outcomes

We evaluated the performance of UM^2Q using our tool described in 4.1. For this, we performed five experiments, and used the scenarios, according to Subsection 4.2, which described 10 sets of providers and the requisites of 5 applications. Each experiment was performed 30 times to reach the normal distribution (Hogendijk and Whiteside, 2011).

In all five experiments, the y-axis represents the average selection time in milliseconds (ms), the rows represent the applications, and the x-axis shows each experiment. The first experiment uses the ten provider sets described in Section 4.2, illustrated by Figure 3. Availability, response time, and cost requirements were not modified during the experiment, maintaining the same value across all sets. Figure 3 shows that increasing the number of providers influences the average selection time as it increases the number of services that meet application requirements. Also, we can observe that the number of microservices also affects the average selection time of providers.

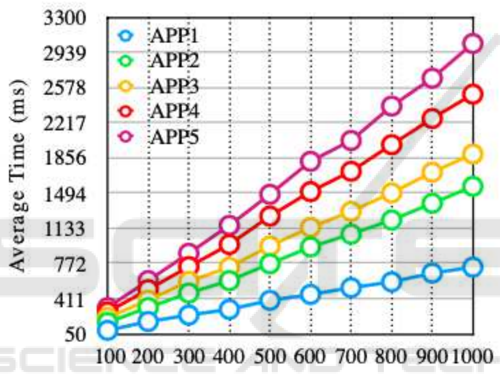


Figure 3: The Set of Providers Experiment.

Figure 4 presents the other four experiments, which were performed using a set of 1000 providers, and each application was set to 5 different values (Figure 4a). The lowest one, 90%, indicates that all provider services must have this minimum value to meet application requirements, and the highest one, 98%, indicates that only cloud services with values between 98 and 100 meet this application requirement, which is in agreement with Eqs. 1 and 2. We set response time and cost requirements to values that can be met by most providers. The results show that the average time decreases with increasing availability requirement value in each application.

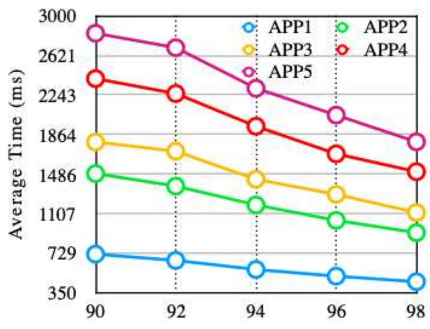
Figure 4b shows the experiment in which we configured each application with five response time values. Thus, the lower the response time value, the fewer the cloud services meet the requirement. We set availability and cost requirements to values that can be achieved by most providers. The graph in Figure 4b shows the selection time increases with the increase of the response time requirement in every application. We created budget classes for the experiment in Figure 4c, in which each class has different budget values for each application, but all applications have the same value per service in each class. The results show that the average selection time increases as the budget class increases. After a certain amount, the average selection time is stabilized, because the ceiling has been reached and even if the budget increases, the number of services to be selected will not increase.

Figure 4d shows the experiment in which we vary the three requirements, in a manner that when the availability requirement increases, the cost and response time requirements decrease. We can observe that the average selection time in the Figure 4d decreases with the requirements restriction increase. This outcome shows that the requirements restriction increase reduces the number of providers that meet all requirements and that the average selection time tends to stabilize because the budget value is higher than the cloud cost value.

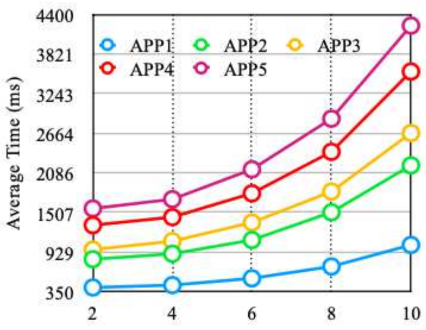
5 RELATED WORK

Several works address the cloud selection problem, but each of them worries about different issues because there are several open ones. In this work, we describe some research works that lead with the selection problem from the user perspective. For this, we describe a summary of these works characteristics in Table 1, which contains seven columns. The first column presents the work reference. The second, third, and fourth columns refer to a taxonomy to identify the work focus that deals with the cloud providers selection regarding the number of providers and services used by them in each phase, which we named providers and services granularity (PSG). The second column shows the number of providers in the selection process, the third one shows the number of selected providers, and the fourth one indicates the granularity per selected provider. Furthermore, the fifth column presents the methods used to solve the selection problem, the sixth one shows whether or not the user defines the requirements threshold, and in the last one presents the goals of the work. In Table 1, n indicates the number of providers in the selection process, which must be greater than 1; while, m indicates the number of selected providers and it must be greater than 1 and lower than or equal to n.

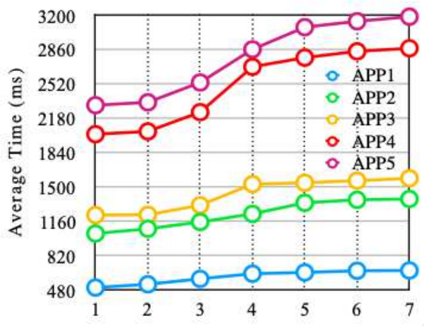
Note in Table 1 that the first four works use a single cloud provider and address the service composition problem. Chen et al., 2016, in the first table line, studied dependency-aware service composition considering multiple QoS attribute to maximize the



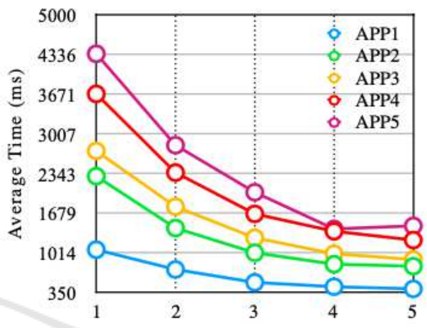
(a) Availability.



(b) Response Time.



(c) Budget Class.



(d) Requirements Class.

Figure 4: Experiments for UM^2Q.

profit but they did not discover the services. While Hongzhen et al., 2016, in the second table line, proposed a strategy of cloud service composition evolution based on QoS, and the QoS values are calculated based on a four-structure model. Liu et al., 2016 proposed an approach for QoS-aware cloud service compostion that allows the user to define the QoS constraints and its preferences. Since microservices are composed of various services, we can consider them as a service composition. Therefore, UM^2Q differs from the Liu et al., 2016 because we address several service composition in parallel, one for each microservice, and we select a service composition among every service composition of all available providers to host a microservice. In the fourth table line, Seghir and Khababa, 2016 also address QoS-aware cloud service, and this work differs both from the Liu et al., 2016 and UM^2Q because of the method used to solve the problem.

We can observe in the fifth and sixth table lines, that the works of Ding et al., 2017 and Jatoth et al., 2018 address the service ranking. They differ from one another by the method used to rank the services. Besides, they differ from UM^2Q because they do not select various services in multiple providers. In the seventh and eighth table lines, Thomas and Chandrasekaran, 2017 and Panda et al., 2018 use cloud federation to execute a user requisition that cannot be executed for selected providers, which makes them only use multiple clouds when necessary.

Lastly, in the ninth, tenth, and eleventh table lines, Sousa et al., 2016, Bharath Bhushan and Pradeep Reddy, 2016, and Mezni and Sellami, 2017 use multiple clouds in both the selection and run-time processes as UM^2Q, but they select services over various clouds while UM^2Q selects microservices. In addition, these works do not allow the user to define the requirements thresholds. Further, UM^2Q is more complex and more flexible, and it has a low communication cost because microservices are independent.

Besides, this paper is part of the PacificCloud project (Carvalho et al., 2018b), which addresses the selection of cloud providers in multi-cloud environments, where native cloud applications are composed of components/modules hosted on different cloud providers/services. In multi-cloud scenarios, choosing an optimal configuration for each application component is challenging once we should considered several requirements, such as performance, availability, or even financial costs. We can point out that these requirements act as selection criteria, and the native cloud applications have their multiple functionalities provided by independent microservices. We investigated different approaches to select

Table 1: Summary of the related work characteristics.

Related Work	Characteristics					
	PSG			Method	user-defined Threshold	Goals
	(1)	(2)	(3)			
(Chen et al., 2016)	1	1	Service Composition	Pareto set model, vector Ordinal Optimization	no	QoS dependency-aware service composition considering multiple QoS atribute
(Hongzhen et al., 2016)	1	1	Service Composition	Hybrid particle swarm	no	cloud service composition evolution
(Liu et al., 2016)	1	1	Service Composition	SLO	yes	Sequential cloud service composition
(Seghir and Khababa, 2016)	1	1	Service Composition	Hybrid GA	yes	QoS-aware cloud service composition
(Ding et al., 2017)	n	1	Service	CSRP	no	Ranking of top-k similar Service
(Jatoth et al., 2018)	n	1	Service	Grey Technique, TOPSIS, AHP	no	Ranking of Service
(Thomas and Chandrasekaran, 2017)	n	m	Resources	TOPSIS, AHP	no	Selection of cloud federation provider to execute a user requisition
(Panda et al., 2018)	n	m	Tasks	Defined by Authors	no	Independent task scheduling in cloud federation
(Sousa et al., 2016)	n	m	Service	Defined by Authors	no	Selection and configuration of multi-cloud for microservices-based applications
(Bharath Bhushan and Pradeep Reddy, 2016)	n	m	Service	PROMETHEE	no	Selection QoS aware services for a service Composition
(Mezni and Sellami, 2017)	n	m	Service	FCA	no	Service composition in multi-cloud that focus on the communication cost
UM^2Q	n	m	Microservices	SAW, Defined by Authors	yes	Multi-cloud selection to host the application microservices

PSG-Providers and Services Granularity, (1) Provider Granularity in the Selection Process, (2) Provider Granularity in the Selection Result, (3) Service Granularity per Provider.

the best set of cloud providers to host each one of the applications' microservices according to software architects' requirements.

The first approach uses dynamic programming in (Carvalho et al., 2018a) and the second ones uses a greedy algorithm in (Carvalho et al., 2019). Finally, in this paper we propose a new approach that uses a multi-criteria strategy where each of the cloud architect provides budget quotas for a set of application microservices. In addition, these quotas must be honored when selecting the available cloud providers.

These three papers follow the same methodology. We established a set of scenarios where we change a set of parameters including (i) the number of microservices that represents each application, (ii) the requirements of each microservice, (iii) the number of available cloud providers or (iv) the number of services offered by each cloud provider. We then checked the overall performance of each solution according to several parameters configuration. In addition, all three papers have one similarity: a software architect must define boundary values for three application properties: availability, response time, and expected budget. The software architect also defines weights among these attributes according to the application priorities (for instance, response time may be more important than availability). The solutions in these articles should select the combinations of services among the available cloud providers that best meet the application's microservices. For each combination, we compute a score according to the software architect's requirements. However, the proposed approach differs from the previous ones as following:

- Instead of setting a global budget quota for the entire application, in the proposed approach, we adopt another strategy where each microservice has its own budget. Once the budget quota is not a global requirement for the whole application anymore, the selection of the best cloud provider for a single microservice depends only on the selection of candidates to host the microservices of each application. This new strategy allows us to filter cloud providers that do not fit the microservices budget independently of any global procedure. It also decreases the search space among the cloud providers, which results in faster responses in the selection process.
- The solutions proposed in Carvalho et al., 2018a and Carvalho et al., 2019 are mapped into the multi-choice knapsack problem and then implemented using a dynamic algorithm and a greedy algorithm, respectively. Both strategies impose an interdependence among the application microservices. the proposed approach does not follow these strategies since a provider is selected for each microservice independently from the other microservices (Sections 3.2 and 3.3). In this way, algorithms in the three approaches are different and share different scenario sizes. The approach that uses a dynamic algorithm imposes limits in the size of the entry (number of microservices in the application and number of available providers), but always selects the best solution (Section 3 in Carvalho et al., 2018a). The greedy algorithm version accepts a larger input and selects a viable solution, but not necessarily the best one (Section III in Carvalho et al., 2019). Nevertheless, the proposed approach accepts large entries and provides the best solution for each microservice based on the definition of the budget quota for each microservice defined by the software architect (Section 4).

6 CONCLUSION

Multi-cloud has stirred the interest of both the academy and the industry to mitigate vendor lock-in as well as to profit from the advantages offered by cloud computing. One of the challenges is that many cloud providers offer many services with the same functionality but with different capabilities, turning the provider selection into a complex task.

PacificClouds aims to manage the deployment and execution of the microservice-based applications in multi-cloud environments. For this, PacificClouds needs to select providers to host the application microservices. Therefore, in this work we propose UM^2Q to select cloud providers to host microservices. The outcomes obtained in the experiments show that the application requirements, the number of application microservices and the number of providers influence the provider's selection time. The results also indicate UM^2Q viability, since the selection time was satisfactory in any scenario, even for extreme values.

One limitation of the UM^2Q is the difficulty of the software architect to determine a budget quota for each microservice of an application. Another limitation of UM^2Q is that it is specific to applications based on microservices that have low communication costs. In regards to these limitation, as future work, we intend to investigate two possible solutions. One to develop a selection process in which the user does not have to determine a budget quota for each microservice and compare the two approaches, and another in the monitoring phase, in which the microservices are already deployed, verifying the expenses for each microservice, resizing the quotas when needed and making a new providers selection to migrate the microservices involved in this process.

In addition, as future work, we intend to implement a recommendation system to suggest better settings than those requested by the user, which would be available with a little more investment.

REFERENCES

Bharath Bhushan, S. and Pradeep Reddy, C. H. (2016). A Qos aware cloud service composition algorithm for geo-distributed multi cloud domain. *International Journal of Intelligent Engineering and Systems*, 9(4):147–156.

Carvalho, J., Vieira, D., and Trinta, F. (2018a). Dynamic Selecting Approach for Multi-cloud Providers. In Luo, M. and Zhang, L.-J., editors, *Cloud Computing – CLOUD 2018*, pages 37–51, Cham. Springer International Publishing.

Carvalho, J., Vieira, D., and Trinta, F. (2019). Greedy Multi-cloud Selection Approach to Deploy an Application Based on Microservices. In *PDP 2019*.

Carvalho, J. O. D., Trinta, F., and Vieira, D. (2018b). PacificClouds : A Flexible MicroServices based Architecture for Interoperability in Multi-Cloud Environments. In *CLOSER 2018*.

Chen, Y., Huang, J., Lin, C., and Shen, X. (2016). Multi-Objective Service Composition with QoS Dependencies. *IEEE Transactions on Cloud Computing*, 7161(c):1–1.

Ding, S., Wang, Z., Wu, D., and Olson, D. L. (2017). Utilizing customer satisfaction in ranking prediction for personalized cloud service selection. *Decision Support Systems*, 93:1–10.

Hayyolalam, V. and Pourhaji Kazem, A. A. (2018). A systematic literature review on QoS-aware service composition and selection in cloud environment. *Journal of Network and Computer Applications*, 110(February):52–74.

Hogendijk, J. and Whiteside, a. E. S. D. (2011). *Sources and Studies in the History of Mathematics and Physical Sciences*. Springer US.

Hongzhen, X., Limin, L., Dehua, X., and Yanqin, L. (2016). Evolution of Service Composition Based on Qos under the Cloud Computing Environment. *Proceedings of ICOACS 2016*, 2016:66–69.

Jatoth, C., Gangadharan, G., Fiore, U., and Computing, R. B. (2018). SELCLOUD: a hybrid multi-criteria decision-making model for selection of cloud services. *Soft Computing*, pages 1–15.

Liu, H., Xu, D., and Miao, H. K. (2011). Ant colony optimization based service flow scheduling with various QoS requirements in cloud computing. *Proceedings - 1st ACIS International Symposium on Software and Network Engineering*, pages 53–58.

Liu, Z. Z., Chu, D. H., Song, C., Xue, X., and Lu, B. Y. (2016). Social learning optimization (SLO) algorithm paradigm and its application in QoS-aware cloud service composition. *Information Sciences*, 326:315–333.

Mezni, H. and Sellami, M. (2017). Multi-cloud service composition using Formal Concept Analysis. *Journal of Systems and Software*, 134:138–152.

Panda, S. K., Pande, S. K., and Das, S. (2018). Task Partitioning Scheduling Algorithms for Heterogeneous Multi-Cloud Environment. *Arabian Journal for Science and Engineering*, 43(2):913–933.

Petcu, D. (2014). Portability in Clouds : Approaches and Research Opportunities, Scalable Computing : Practice and Experience. 15(3):251–270.

Seghir, F. and Khababa, A. (2016). A hybrid approach using genetic and fruit fly optimization algorithms for QoS-aware cloud service composition. *Journal of Intelligent Manufacturing*, pages 1–20.

Somu, N., Gauthama, G. R., Kirthivasan, K., and Shankar, S. S. (2018). A trust centric optimal service ranking approach for cloud service selection. *Future Generation Computer Systems*, 86:234–252.

Sousa, G., Rudametkin, W., and Duchien, L. (2016). Automated Setup of Multi-Cloud Environments for Microservices-Based Applications. *9th IEEE International Conference on Cloud Computing*.

Tang, M., Dai, X., Liu, J., and Chen, J. (2017). Towards a trust evaluation middleware for cloud service selection. *Future Generation Computer Systems*, 74:302–312.

Thomas, M. V. and Chandrasekaran, K. (2017). Dynamic partner selection in Cloud Federation for ensuring the quality of service for cloud consumers. *International Journal of Modeling, Simulation, and Scientific Computing*, 08(03):1750036.

Zheng, H., Feng, Y., and Tan, J. (2016). A fuzzy QoS-aware resource service selection considering design preference in cloud manufacturing system. *International Journal of Advanced Manufacturing Technology*, 84(1-4):371–379.

Zhou, J., Yao, X., Lin, Y., Chan, F. T., and Li, Y. (2018). An adaptive multi-population differential artificial bee colony algorithm for many-objective service composition in cloud manufacturing. *Information Sciences*, 456:50–82.