A Multi-cloud Parallel Selection Approach for Unlinked Microservice Mapped to Budget’s Quota: The $PUM^2Q$

Juliana Carvalho$^1$, Fernando Trinta$^2$, and Dario Vieira$^3$

$^1$ University Federal of Piauí - UFPI, Picos, Brazil  
julianaoc@ufpi.edu.br
$^2$ University Federal of Ceará - UFC, Fortaleza, Brazil  
fernando.trinta@dc.ufc.br
$^3$ EFREI-Paris, Paris, France  
dario.vieira@efrei.fr

Abstract. The world is facing a complicated moment in which social isolation is necessary. Therefore, to minimize the problems of companies, remote work is being widely adopted, which is only possible because of existing technologies, including cloud computing. Choosing the providers to host the business applications is a complex task, as there are many providers and most of them offer various services with the same functionality and different capabilities. Thus, in this paper, we propose an approach, called $PUM^2Q$, for selecting providers to host a distributed application based on microservices that have little communication between them. $PUM^2Q$ is a provider selection approach based on multi-criteria, and it copes with the needs of microservices individually and in parallel. The proposed approach extends our previous one, $UM^2Q$, and should be incorporated by PacificClouds. Besides, we carry out a performance evaluation by varying the number of requirements, microservices, and providers. We also compare $PUM^2Q$ and $UM^2Q$. The results presented by $PUM^2Q$ are better than those given by $UM^2Q$, showing not only its viability but also expanding the number of approaches adopted by PacificClouds. As a result, $PUM^2Q$ making the tasks of the software architect, who is the user of PacificClouds, more flexible.

Keywords: Multi-criteria decision-making method · Microservices · Multi-cloud parallel selection

1 Introduction

Despite existing for a long time, remote work was not widely adopted. In the face of a pandemic in which social isolation is essential, many companies have started using it. Many remote jobs are only possible due to the existence of several technologies, and one of them being cloud computing. According to Flexera$^1$,
30% of companies are adopting more cloud computing than predicted before the pandemic. Flexera\(^2\) also points out that 84% of companies with more than 1,000 employees have as a strategy for the use of multiple clouds. Given this context, we can see the importance of cloud computing and, more specifically, of a multi-cloud environment in the current and future scenario.

The use of multiple clouds brings many advantages, as it consolidates the benefits obtained by cloud computing, allowing the user to choose the services that best meet their needs regardless of the provider. Effectively, using multiple providers and multiple cloud services per provider to compose a distributed application has several challenges, including those described by Flexera (see Footnote 2), such as cloud cost management, governance, security, compliance, lack of resources/expertise, multiple clouds management and cloud migration.

In this paper, we propose an approach for selecting multiple providers to host an application based on microservices from the software architect’s perspective. The applications considered in this work have little interaction between microservices so we disregarded them. The proposed approach, named \(PUM^2Q\), selects providers considering the requirements of availability, response time and cost. \(PUM^2Q\) treats the provider selection for a microservice regardless of another microservice. For this, a software architect needs to define an application budget quota for each microservice. The proposed approach extends the work published in [4], in which we propose the \(UM^2Q\) approach. The two approaches differ because the one proposed in this work parallels each provider selection for each microservice. Thus, \(PUM^2Q\) increases the number of scenarios served by \(UM^2Q\), as the selection time decreases, and it supports scenarios with a more significant amount of microservices, providers and services per provider. To demonstrate this, we propose a performance evaluation as well as a comparison between the two proposals described in Sect. 5.

PacificClouds [5] is a platform that aims to manage the deployment and execution of an application based on microservices distributed in a multi-cloud environment. 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 provide 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.

PacificClouds intends to incorporate several provider selection approaches in order to address a higher number of scenarios and, consequently, the software architect’s needs. We propose other selection approaches besides the one described in [4], which meet different scenarios. The method proposed in [2] obtains the best solution, but it is limited in the input size, that is, in the number of microservices of an application and the amount of available providers. The approach proposed in [3] obtains a viable solution and accepts a large entry, that is, an application with a large number of microservices and providers.

\(^2\)https://www.flexera.com/blog/cloud/2019/02/cloud-computing-trends-2019-state-of-the-cloud-survey/.
The $UM^2Q$ introduced in [4] gets an optimal local solution and accepts a large entry, but smaller than the proposal in [3]. The purpose of $PUM^2Q$ is to take advantage of $UM^2Q$ and expand the size of the entrance. Therefore, $PUM^2Q$ is an approach that contributes to increase the number of scenarios served by PacificClouds.

Once this approach focuses on applications based on microservices distribute in a multi-cloud environment, we adopted the definition for multi-cloud [5], which is a delivery model where there is no agreement between the providers involved. We use the definition of microservices described in [5], followed by an approach example in Sect. 2. In Sect. 3 we present a formulation for the model; and, in Sect. 4 the $PUM^2Q$ approach description.

Other works in the literature deal with the selection process in cloud computing, but as described in [4], they focus on different aspects. In [4], as presented in Sect. 6, we describe some works that focus on the user’s perspective and propose a taxonomy called providers and services granularity (PSG), which classifies the papers according to the number of providers and services used both in the selection phase and in the application. Further, we present other works that use parallel approaches to address the selection of cloud providers, and we also compared them to $PUM^2Q$.

According to the aspects described in this section, the main contributions of this work are as follows:

1. We propose a provider selection process for an application microservice regardless of another microservice in the same application. For this, a software architect must define an application budget quota for each microservice;
2. We propose a parallel selection of providers to host an application’s microservices;
3. We describe a formal description of the proposed approach;
4. We present a tool that implements the proposed approach;
5. We assess a performance evaluation to verify the $PUM^2Q$ feasibility. Furthermore, we compare $PUM^2Q$ and $UM^2Q$.

2 Example for $PUM^2Q$

The main objective of $PUM^2Q$ is to choose providers in parallel to host an application based on microservices from the software architect’s perspective, in a manner that, the approach selects a single provider for each microservice. For this, a software architect needs to provide $PUM^2Q$ with the requirements of each application’s microservice and the available providers’ capabilities. In this section, we present a simplified example for $PUM^2Q$.

The example shows an application with three microservices represented by MS1, MS2 and MS3, illustrated in Fig. 1a. MS1 requires three cloud services, represented by S11, S21 and S31. Figure 1a also shows the services of the provider Cloud1, which has 6 services represented by CS11, CS21, CS31, CS41, CS51 and CS61. Figures 1b, 1c and 1d show the selection of candidate services from the provider Cloud1 that meet each cloud services required to compose MS1.
A Multi-cloud Parallel Selection Approach

(a) Microservices’ Application and Available Provider.
(b) Candidate Services for S11 from MS1.
(c) Candidate Services for S21 from MS1.
(d) Candidate Services for S31 from MS1.

Fig. 1. Examples for candidate services for $PUM^2Q$.

Figure 2 shows the candidate combinations to serve MS1, according the provider Cloud1 capabilities. Next, $PUM^2Q$ selects the provider combination Cloud1 for MS1 that has the highest score. The approach calculates the score by the weighted average between the values for availability, response time and cost and the weight of each one. The weight represents the priority of the requirement. $PUM^2Q$ repeats the process on all available providers for MS1. In the end, there is a set of candidate combinations for MS1, one for each provider. Again, the approach selects the best candidate among providers by looking at their score. $PUM^2Q$ performs the entire process for all microservices in parallel.

Fig. 2. Candidate combinations for $PUM^2Q$.

Now, let’s assume that the budget defined for the application is 12 and that the budget quota for MS1 is 4, for MS2 is 5 and for MS3 is 3. Figure 3 shows the set of candidate combinations for each application microservice containing the score, the cost and the provider to which it belongs. We can observe that combination 3 for MS1 has the highest score, and it does not exceed the budget’s quota for MS1. We can also notice that combination 3 for MS2 has the highest score, but it does exceed the budget’s quota. So $PUM^2Q$ must select combination 2 for MS2, as shown in Fig. 4. Its happens similarly with MS3 and the approach selects combination 2 to fit the budget, even though combination 3 has the highest score.
3 Formulation

The proposed selection approach refers to the cloud providers selection from the software architect’s perspective before an application deployment. The software architect must define their requirements for an application, and set a budget’ quota for each application microservice. We consider that the provider’s capabilities are available since this is an other challenge and we do not treat in this paper. In this section, we present a formal description for $PUM^2Q$ through definitions, and the optimization of availability, response time and cost requirements.

As we can observe in Sect. 2, the required cloud services to compose a microservice have different requirements and the providers offer many services with the same functionalities but different capabilities. Even though our approach can handle as many requirements as needed, in this paper, we consider three user requirements: (i) response time (execution time + communication delay), (ii) availability, and (iii) application execution cost, but other requirements can be included in our model easily. We use three user’s requirements that are enough to understand our proposed selection process. However, the software architect can define several other user requirements without significant changes in the code.

According to the previously described providers’ selection process, and as in [4], we propose:

| Microservice | Combination | Quota | Score | Cost | Provider |
|--------------|-------------|-------|-------|------|----------|
| MS1          | 1           | 4     | 0.6   | 2    | P1       |
|               | 2           |       | 0.7   | 3    | P2       |
|               | 3           |       | 0.8   | 4    | P3       |
| MS2          | 1           | 5     | 0.5   | 2    | P5       |
|               | 2           |       | 0.7   | 4    | P4       |
|               | 3           |       | 0.8   | 7    | P3       |
| MS3          | 1           | 3     | 0.6   | 1    | P2       |
|               | 2           |       | 0.7   | 3    | P5       |
|               | 3           |       | 0.8   | 8    | P1       |

**Fig. 3.** All candidate combinations for $PUM^2Q$.

**Fig. 4.** Combinations selection for $PUM^2Q$. 
– 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.

– Definition 2: **cloud services class** - as $SC_{l} = \{S_{l1}, S_{l2}, \ldots, S_{lo}\}$, in which $S_{l1}$, $S_{l2}$, $\ldots$, $S_{lo}$ 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 $SC_{k1}$, $SC_{k2}$, $\ldots$, $SC_{kp}$ are services classes.

– Definition 4: **cloud provider set** - as $CP = \{SP_{1}, SP_{2}, \ldots, SP_{q}\}$, in which $SP_{1}$, $SP_{2}$, $\ldots$, $SP_{q}$ are services providers.

– Definition 5: **microservice model** - as $MS_{i} = \{S_{ki1}, S_{ki2}, \ldots, S_{kir}\}$, in which $S_{ki1}$, $S_{ki2}$, $\ldots$, $S_{kir}$ 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, $1 \leq k_{r} \leq o$, 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.

All definitions described for the cloud selection process must follow the model of definition 1. Example: Definition X is modelled as $X(X_{rt}, X_{a}, X_{c})$.

### 3.1 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 to be defined as the lowest availability value among its microservices, and it must be greater than or equal to $MinAvbty$. Also, in Eq. 2, $MS_{i}.a$ represents the microservice availability $i$, which is assigned to be the lowest availability value among cloud services that compose $MS_{i}$ and it must be greater than or equal to $MinAvbty$. We represent the cloud availability for a service by $S_{ij}.a$ in Eq. 2, which is the service $j$ from provider $K$ for the microservice $i$.

\[
AP.a = \min_{1 \leq i \leq t} (MS_{i}.a) \geq MinAvbty, \forall MS_{i} \in AP
\]  

\[
MS_{i}.a = \min_{1 \leq j \leq r} (S_{ij}.a) \geq MinAvbty,
\forall S_{ij} \mid S_{ij} \in MS_{i}, MS_{i} \in AP, 1 \leq i \leq t
\]

### 3.2 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 $\text{MaxRT}$ is the maximum execution time threshold ($\text{MaxExecTime}$) plus the maximum delay threshold ($\text{MaxDelay}$) defined by the user, as in Eq. 3. In Eq. 4, we define $\text{AP}rt$ 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 $\text{MaxRT}$. In addition, in Eq. 5, $MS_{i,rt}$ represents the response time of microservice $i$, which is assigned to be the highest response time value among their services and that must not be less than $\text{MaxRT}$. We represent the response time for a service by $S_{ij}^{k,rt}$ in Eq. 5, which is the service $j$ from provider $k$ for the microservice $i$.

$$MaxRT = \text{MaxExecTime} + \text{MaxDelay}$$  \hspace{1cm} (3)

$$\text{AP}rt = \max_{1 \leq i \leq t} (MS_{i,rt}) \leq \text{MaxRT}, \forall MS_i | MS_i \in AP$$  \hspace{1cm} (4)

$$MS_{i,rt} = \max_{1 \leq j \leq r} (S_{ij}^{k,rt}) \leq \text{MaxRT},$$

$$\forall S_{ij}^{k} | S_{ij}^{k} \in MS_i, MS_i \in AP, 1 \leq i \leq t$$  \hspace{1cm} (5)

### 3.3 Cost Requirement

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

$$\text{AP}c = \sum_{i=1}^{t} MS_{i,c} \leq \text{Budget}, \forall MS_i | MS_i \in AP$$  \hspace{1cm} (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.

### 4 $PUM^2Q$ Approach

We can observe in the previous sections that there are various available cloud providers, and each of them offers several services. An application can consist of many microservices, and each of them must require several cloud services to meet its tasks’ requirements. Therefore, selecting cloud providers to host a distributed application in multiple clouds is complex. In this section, we propose $PUM^2Q$, a multi-cloud parallel selection approach to host the unlinked microservices mapped to quota’s budget. The proposed parallel selection model refers to the cloud providers selection from the software architect’s perspective before an application deployment.
In PUM2Q, as in UM2Q [4], we select multi-cloud providers to host each application microservice and each one of them is hosted by a single provider, considering only the microservice and software architect requirements. PUM2Q hosts a microservice in a single provider so that its tasks do not exchange information among services from different providers. This approach presents three selection levels as in UM2Q. In both, UM2Q and PUM2Q approaches, we execute the levels sequentially, but while in UM2Q, the provider selection process for each microservice is sequential, in PUM2Q is in parallel. The three levels will be described next.

4.1 First Level: Determine Candidate Services for a Microservice

We select the services of each provider that meet all software architect requirements, which results in a candidate services set from each provider for one microservice. Next, we rank all candidate services in each provider. We use the Simple Additive Weighting (SAW) technique as in [16], which has two phases: scaling and weighting.

**Scaling Phase.** First, a matrix \( R = (R_{ij}; 1 \leq i \leq n; 1 \leq j \leq 3) \) is built by merging the requirement vectors of all candidate services. For this, the user requirements are numbered from 1 to 3, where 1 means availability, 2 means response time, and 3 means 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, \( R_{j}^{Max} = \text{Max}(R_{ij}), R_{j}^{Min} = \text{Min}(R_{ij}), 1 \leq i \leq n. \)

**Negative:** the higher the value, the lower the quality.

\[
V_{ij} = \begin{cases} 
\frac{R_{j}^{Max} - R_{ij}}{R_{j}^{Max} - R_{j}^{Min}} & \text{if } R_{j}^{Max} - R_{j}^{Min} \neq 0 \\
1 & \text{if } R_{j}^{Max} - R_{j}^{Min} = 0 
\end{cases} \tag{7}
\]

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

\[
V_{ij} = \begin{cases} 
\frac{R_{ij} - R_{j}^{Min}}{R_{j}^{Max} - R_{j}^{Min}} & \text{if } R_{j}^{Max} - R_{j}^{Min} \neq 0 \\
1 & \text{if } R_{j}^{Max} - R_{j}^{Min} = 0 
\end{cases} \tag{8}
\]

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

\[
\text{Score}(S_i) = \sum_{j=1}^{3} (V_{ij} \times W_j) \mid W_j \in [0, 1], \sum_{j=1}^{3} W_j = 1 \tag{9}
\]

At the end of the first level, we have a set of all candidate services for one microservice from all available providers.
4.2 Second Level: Determine a Candidate Combination from All Providers

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 ($Budget \times Quota_i$). 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 sum of all ($Budget \times Quota_i$) must be equal to the Application 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 $i$.

$$MS_i.c \leq Budget \times Quota_i, \forall MS_i \in AP \quad (10)$$

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

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

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 $(S_{ij}^1, ..., S_{ij}^r)$ in Eq. 13. Each element of the candidate combination $S_{ij}^k$ represents the candidate service $j$ of provider $k$ to the microservice $i$, in which $1 \leq j \leq r$ 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 $(Budget \times Quota_i)$ as shown in Eq. 13, which is in accordance to Eqs. 10, 11, and 12. Each microservice has a maximum $r$ services.

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

Next, we must choose a combination among candidate services combinations in each provider for each microservice. For this, our approach 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 defined by a software architect, but according to his priorities.

At the end of the second level, there must be a candidate provider set \((CSP_i)\) for the \(MS_i\) of an application, as shown in Eq. 15. \(SP_{ki}\) represents the candidate provider \(k\) for 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.

\[
CSP_i = \{SP_{1i}, SP_{2i}, \ldots, SP_{qi,i}\} \mid 1 \leq i \leq t, \quad t = \text{size of}(AP), q_i = \text{size of}(CSP_i)
\]  

(15)

### 4.3 Third Level: Determine a Combination for a Microservice

We select one provider for a 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 \(k\).

\[
SP_{ki.c} = \sum_{j=1}^{r}(S_{ij}^{kj}.c) \mid S_{ij}^{kj} \in SP_{ki}, SP_{ki} \in CSP_i, \quad 1 \leq k \leq q_i, 1 \leq i \leq t
\]  

(16)

Next, we check the CSP from the Second Level for the 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 \(i\).

\[
Max(MS_i.c) = \max_{1 \leq k \leq s_i} (SP_{ki.c}) \mid \forall SP_{ki} \in CSP_i, \quad 1 \leq i \leq t, q_i = \text{size of}(CSP_i)
\]  

(17)

\[
Min(MS_i.c) = \min_{1 \leq k \leq s_i} (SP_{ki.c}) \mid \forall SP_{ki} \in CSP_i, \quad 1 \leq i \leq t, q_i = \text{size of}(CSP_i)
\]  

(18)

\[
Average(MS_i.c) = \frac{Max(MS_i.c) + Min(MS_i.c)}{2}
\]  

(19)

If there is more than one provider with the same average execution cost, we select the one with highest availability or performance, observing the priority defined by a system architect. 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 one 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.

$$SP_{ki}.a = \min_{1 \leq j \leq r} (S_{ij}^{k}.a) \mid S_{ij}^{k} \in SP_{ki}, SP_{ki} \in CSP_{i}, 1 \leq k \leq q_{i}, 1 \leq i \leq t$$ \hspace{1cm} (20)

$$SP_{ki}.rt = \max_{1 \leq j_{x} \leq r} (S_{ij_{x}}^{k}.rt) \mid S_{ij_{x}}^{k} \in SP_{ki}, SP_{ki} \in CSP_{i}, 1 \leq k \leq q_{i}, 1 \leq i \leq t$$ \hspace{1cm} (21)

$$\text{Max}(MS_{i}.a) = \max_{1 \leq k \leq s_{i}} (SP_{k}.a) \mid \forall SP_{k} \in CSP_{i}$$ \hspace{1cm} (22)

$$\text{Max}(MS_{i}.rt) = \min_{1 \leq k \leq s_{i}} (SP_{k}.rt) \mid \forall SP_{k} \in CSP_{i}$$ \hspace{1cm} (23)

We execute the three levels sequentially for a microservice and in parallel for the microservices.

5 Performance Assessment

In this section, we describe the scenarios configuration, the tool, the experiments, and the outcomes of $PUM^{2}Q$ and $UM^{2}Q$, and we present an analysis of the results.

5.1 Tool Description

We developed a tool to evaluate the $PUM^{2}Q$ and $UM^{2}Q$ 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 and the other has the application requirements. Figure 5 shows a JSON example for application and provider. In Fig. 5a, we can observe the definition of the requirements ‘values, the budget’s quota and the microservice capabilities for an application. In Fig. 5b, we can notice the definition of classes, services and the capabilities of a microservice for a provider. 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 Sect. 4, $PUM^{2}Q$ 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.

5.2 Setting Scenarios

According to [8], there are few available datasets in the research domain. In this manner, in 50% of the works referenced by [8], the authors use a dataset configured randomly. So, we randomly configured eight sets of providers, and each provider has three services classes: compute, storage, and database. The first four provider sets differ from one another by the number of providers, and the last four ones by number of services per provider. We define these numbers of providers per set because we can demonstrate through them the flexibility of scenarios for the proposed solutions. Table 1 presents the number of sets, the amount of providers of each set, the number of services by class, the amount of services by providers and the number of services in each set.

Also, we configured the requirements for microservices-based applications. Each microservice contains one service of each class: compute, storage, and database. We set each application by microservice. Availability, response time,
and cost requirements vary depending on the type of assessment that should be performed. Table 2 illustrates all applications used in the experiments. We define these applications to show the flexibility of $PUM^2Q$.

Table 1. Set of providers.

| Number of sets | Number of providers by sets | Number of services by class | Number of services by provider | Number of services by set |
|----------------|-----------------------------|-----------------------------|-------------------------------|--------------------------|
| 4              | 200                         | 30                          | 90                            | 18000                    |
|                | 300                         |                             |                               | 27000                    |
|                | 400                         |                             |                               | 36000                    |
|                | 500                         |                             |                               | 45000                    |
| 4              | 200                         | 20                          | 60                            | 12000                    |
|                |                             | 30                          | 90                            | 18000                    |
|                |                             | 40                          | 120                           | 24000                    |
|                |                             | 50                          | 150                           | 30000                    |

Table 2. Set of applications.

| Applications | Characteristics | Number of microservice | Number of services by application |
|--------------|----------------|------------------------|----------------------------------|
| App1         |                | 5                      | 15                               |
| App2         |                | 6                      | 18                               |
| App3         |                | 8                      | 24                               |
| App4         |                | 10                     | 30                               |
| App5         |                | 12                     | 36                               |
| App6         |                | 14                     | 42                               |
| App7         |                | 15                     | 45                               |
| App8         |                | 16                     | 48                               |
| App9         |                | 18                     | 54                               |
| App10        |                | 20                     | 60                               |
| App11        |                | 22                     | 66                               |
| App12        |                | 25                     | 75                               |
| App13        |                | 30                     | 90                               |
| App14        |                | 35                     | 105                              |
| App15        |                | 40                     | 120                              |
| App16        |                | 45                     | 135                              |
| App17        |                | 50                     | 150                              |
Each experiment was performed 30 times to reach the normal distribution [9]. We made the experiments on a Xenon processor Quad Core, 4 GB of RAM and a 256 GB SSD.

### 5.3 Experiments for $PUM^2Q$

We evaluated the performance of $PUM^2Q$ using the tool described in Sect. 5.1. For this, we carried out seven experiments which vary the number of providers, number of services per provider, number of microservices per application, varying each requirement individually and then all they at the same time. Thus, we check for the possible variations that may influence the performance. For performing the experiments, we used the scenarios, according to Subsect. 5.2, which described 8 sets of providers and the requisites of 17 applications.

In all seven experiments, the y-axis represents the average selection time in seconds (s), the rows represent the applications, and the x-axis shows each experiment. The first and second experiments use the first and last four providers sets, respectively, described in Sect. 5.2, illustrated by Figs. 6a and 6b. Availability, response time, and cost requirements were not modified during the two experiments, maintaining the same value across all sets. Figures 6a and 6b show that increasing the number of providers and services respectively influence the average selection time. We can notice the increase in the number of services has more considerable influence than the increase in the number of provider.

![Fig. 6. Experiments based on number of providers and services for $PUM^2Q$.](image)

Observing the results of the first two experiments presented in Figs. 6a and 6b, we use the set of providers with 200 providers and 30 services per class of provider, with a total of 90 services per provider, for the other experiments because they represent intermediate values.

The third experiment illustrated in Fig. 7 has an objective observing the behaviour of the approach with the increase of the number of microservice and services by an application. We configure requirements of availability, response
time, and cost with of intermediary values and use the set of providers with 200 providers, the first set of providers in Table 1. The results show that the average selection time is viable for 50 microservices or 150 services by the application using the intermediary values for the requirements.

Fig. 7. The experiment based on microservice by application for $PUM^2 Q$.

Figure 8 presents the other four experiments, which were performed using a set of two hundred providers, and each application was set to four different values. In Fig. 8a), the lowest one, 92%, 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 the increasing availability requirement value in each application.

Figure 8b shows the experiment in which we vary the response time’s values. Thus, the lower the response time value, the fewer the cloud services that meet the requirement. We set availability and cost requirements to values that can be achieved by most providers. Figure 8b shows the selection time increases with the increase of the response time requirement in every application. We created budget classes for the experiment in Fig. 8c, 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.

Figure 8d 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 notice that the average selection time decreases with the requirements restriction increase. This outcome shows that the requirements restriction increase reduces the number of providers that meet all requirements.
5.4 Experiments for $UM^2Q$

We performed the same $PUM^2Q$ experiments with the $UM^2Q$ to compare their response. We described the $UM^2Q$ approach in [4], and the main difference between $PUM^2Q$ and $UM^2Q$ is that the former makes the selection in parallel and the latter in sequence. Thus, we carried out seven experiments and used the scenarios, according to the experiments for $PUM^2Q$.

We performed the seven experiments, the graphics shows in this paper represent the results, and their y-axis represents the average selection time in milliseconds (s), the rows represent the applications, and the x-axis shows each experiment. Figure 9a illustrates the experiment, which we vary the number of providers by set. We can observe that increasing the number of providers influences the average selection time. Figure 9b shows the experiment, in which we vary the number of services by the provider, and the average selection time increases with the growth in the number of services per provider. We can see in Figs. 9a and 9b, that the variation in the number of services has a more considerable influence than in the number of providers.

Figure 10 illustrates the experiment, which varies the number of microservice by application, and we configured the requirements of availability, response...
time and cost with the intermediary values. We can observe that the number of microservices’ influences the average selection time.

Figure 11 presents the other four experiments, which we performed using a set of 200 providers, and set each application to four different values. We can notice that in Fig. 11a, the values start with 92%, and end with 98%. We set response time and cost requirements to values that can be met by most providers. The results show that the average selection time decreases with the increasing availability requirement value in each application.

Figure 11b shows the experiment in which we vary the response time values. We set availability and cost requirements to values that can be achieved by most providers. We can observe in Fig. 11b that the selection time increases with the increase of the response time requirement in every application. In Fig. 11c, we created budget classes for the experiment; each class has different budget values for each application, and all applications have the same value per service. The results show that the average selection time increases as the budget class increases.
In the last experiment, we vary the three requirements in a manner that when the availability requirement increases, the cost and response time requirements decrease, as illustrated by Fig. 11d. We can observe that the average selection time decreases with the requirements restriction increase.

### 5.5 Comparison Between PUM²Q and UM²Q

According to the experiments presented in Subsects. 5.3 and 5.4, we can notice that the applications have the same behaviour. The experiments differ in the aspect that the applications with fewer microservices in the PUM²Q approach have a very close average selection time, while in UM²Q the selection time between these applications is longer. The main difference is in the average selection time in all experiments in a comprehensive way. The PUM²Q approach is 70% faster than the UM²Q approach. The difference between them can be even more significant if performed on machines with a larger number of cores. Therefore, the PUM²Q approach is more flexible than UM²Q, as it allows for great scenarios, in which applications may have a higher number of microservices and services, as well as providers, may have a more substantial amount of classes and services per class.
Several works address the cloud selection problem, but each of them worries about different issues because there are several open ones. In Table 3, which contains seven columns, describe some research works that lead with the selection problem from the user perspective. The first column presents the work

Table 3. Summary of the related work characteristics based on [4].

| Related work | Characteristics |
|--------------|-----------------|
| PSG          | Method          | User-defined threshold | Goals                        |
| (1) (2) (3)  |                 |                            |                             |
| [6]          | 1   1 Service composition | Pareto set model, VOO     | No                          | QoS dependency-aware service composition |
| [10]         | 1   1 Service composition | HPS                       | No                          | Cloud service composition           |
| [13]         | 1   1 Service composition | SLO                       | Yes                         | Sequential service composition     |
| [16]         | 1   1 Service composition | Hybrid GA                 | Yes                         | QoS-aware service composition      |
| [12]         | 1   1 Service composition | PMMAS-CL                  | No                          | The service composition considering service correlation based on the library |
| [19]         | 1   1 Service composition | PDPSO                     | No                          | Cloud service selection            |
| [7]          | n   1 Service    | CSRP                      | No                          | Ranking of top-k similar Service   |
| [11]         | n   1 Service    | Grey Technique, TOPSIS, AHP | No                          | Ranking of Service                 |
| [18]         | n   m Resources  | TOPSIS, AHP               | No                          | Selection of cloud federation provider to execute a user requisition |
| [15]         | n   m Tasks      | Defined by Authors        | No                          | Independent task scheduling in cloud federation |
| [17]         | n   m Service    | Defined by Authors        | No                          | Selection and configuration of multi-cloud for microservices-based applications |
| [1]          | n   m Service    | PROMETHEE                 | No                          | Selection QoS aware services for a service Composition |
| [14]         | n   m Service    | FCA                       | no                          | Service composition in multi-cloud that focus on the communication cost |
| [20]         | n   m Service Composition | MPSaDABC                  | Yes                         | Large-scale service composition for cloud manufacturing |

PSG-Providers and Services Granularity, (1) Provider Granularity in the Selection Process, (2) Provider Granularity in the Selection Result, (3) Service Granularity per Provider
reference, and the second, third, and fourth ones 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) and described in [4]. The second column shows the number of providers in the selection process, the third one shows the number of selected providers, the fourth one indicates the granularity per selected provider, the fifth one 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 3, \( n \) 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 3 that the first four works use a single cloud provider and address the service composition problem. [6], in the first table line, studied dependency-aware service composition considering multiple QoS attributes to maximize the profit but they did not discover the services, while [10], 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. [13] proposed an approach for QoS-aware cloud service composition 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, \( PUM^2Q \) differs from the [13] 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, [16] also addresses QoS-aware cloud service, and this work differs both from the [13] and \( PUM^2Q \) because of the method used to solve the problem.

In 5 and 6’s table lines works also use a single provider, but these works lead with the service composition in parallel. In [12], the authors propose a model to address the QoS correlation problem to optimize the service composition in cloud manufacturing. Besides, they offer an algorithm named Parallel Max-min Ant System based on the case library (PMMA-CL) to streamline the service composition, the algorithm uses ant colony that maintains the divergence of the population and then a learning strategy is adopted simultaneously to accelerate the convergence rate through particular ants based on dynamic database lists named library. [19] proposes an optimization algorithm for cloud service selection using parallel discrete particle swarm optimization (PDPSO), which uses the map/reduce mechanism to perform parallelization. The authors define each group of services as a particle, and each particle is a thread. They evaluated a fixed number of services as well as combinations. The proposed algorithm is efficient for a sequence of services, whereas \( PUM^2Q \) has at least one sequence for each microservice. Also, as \( PUM^2Q \) we deal with a multi-cloud environment, there are several candidate services as well as many candidate combinations.

We can observe in the seventh and eighth table lines, that the works of [7] and [11] 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, [18] and [15] use cloud federation to execute a user requisition that cannot be executed for selected providers, which makes them only use multiple clouds when necessary.

In table line 11, [17] proposes an automated approach for the selection and configuration of cloud providers for multi-cloud microservices-based applications. The approach uses a domain specific language to describe the application’s multi-cloud requirements and the authors provide a systematic method for obtaining proper configurations that comply with the application’s requirements and the cloud providers’ constraints. In table line 12, [1] proposes an algorithm to select a cloud with more number of service files and then we applied PROMETHEE, a multi criteria decision making method that selects the best service based on QoS criteria from optimal cloud combinations. In table line 13, [14] proposes a multi-cloud service composition (MCSC) approach based on Formal Concept Analysis (FCA). The authors use FCA to represent and combine information on multiple clouds. FCA is based on the concept of lattice, which is a powerful mean to classify clouds and services information. The authors first model the multi-cloud environment as a set of formal contexts. Next, they extract and combine candidate clouds from formal concepts. Finally, they select the optimal cloud combination and transform the MCSC into a classical service composition problem. [1, 17], and [14] use multiple clouds in both the selection and execution processes as $PUM^2Q$, but they select services over various clouds while $PUM^2Q$ selects microservices. In addition, these works do not allow the user to define the requirements thresholds. Further, $PUM^2Q$ is more complex and more flexible, and it has a low communication cost because the microservices are independent.

Lastly, in table line 14, [20] propose a multi-population parallel self-adaptive differential artificial bee colony (MPsaDABC) algorithm for composite cloud manufacturing (CMfg) optimal service selection (CCSOS). The proposed algorithm adopts multiple parallel subpopulations, each of which evolves according to different mutation strategies borrowed from the differential evolution (DE) to generate perturbed food sources for foraging bees, and the control parameters of each mutation strategy are adapted independently. Moreover, the size of each subpopulation is dynamically adjusted based on the information derived from the search process. The algorithm proposed focuses on selecting one candidate service from each corresponding candidate service set to form the composite service while ensuring the overall QoS is optimal, whereas $PUM^2Q$ deals with discovery candidate service to service composition. To validate the algorithm MPsaDABC, the authors use candidate service set scales, whereas in $PUM^2Q$ we made the discovery services then took the candidate services. The number of candidate services in $PUM^2Q$ vary according to the number of providers and services per providers.
7 Conclusion

The use of multi-cloud is interesting for both the academy and the industry, since it can help take the advantages offered by cloud computing by using the most suitable services for an application. The services selection is a complex task, because the providers offer many services with the same functionalities and different capabilities.

To facilitate the decision making by a software architect, PacificClouds intends to manage the deploying and executing process a microservice-based application in multi-cloud environments. One of the first steps for PacificClouds is to select providers and their services to host an application microservices. Thus, in this work, we propose $PUM^2Q$, which extends $UM^2Q$ [4]. The $PUM^2Q$ approach selects providers to host the application’s microservices in parallel. The outcomes show that the requirements, the number of microservices and the number of providers and services per provider influence the average selection time. The results also indicate that $PUM^2Q$ is more flexible than $UM^2Q$ because it allows for substantially larger scenarios.

The $PUM^2Q$ has the same limitation of the $UM^2Q$, which is the difficulty of the software architect to determine a budget quota for each application microservice. Besides, 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

1. Bhushan, S.B., Reddy, C.H.P.: A QoS aware cloud service composition algorithm for geo-distributed multi cloud domain. Int. J. Intell. Eng. Syst. 9(4), 147–156 (2016). https://doi.org/10.22266/ijies2016.1231.16
2. Carvalho, J., Vieira, D., Trinta, F.: Dynamic selecting approach for multi-cloud providers. In: Luo, M., Zhang, L.-J. (eds.) CLOUD 2018. LNCS, vol. 10967, pp. 37–51. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94295-7_3
3. Carvalho, J., Vieira, D., Trinta, F.: Greedy multi-cloud selection approach to deploy an application based on microservices. In: PDP 2019 (2019). https://doi.org/10.1109/PDP.2019.00021
4. Carvalho, J., Vieira, D., Trinta, F.: UM2Q: multi-cloud selection model based on multi-criteria to deploy a distributed microservice-based application, pp. 56–68 (2020). https://doi.org/10.5220/0009338200560068
5. de Carvalho, J.O., Trinta, F., Vieira, D.: PacificClouds: a flexible MicroServices based architecture for interoperability in multi-cloud environments. In: CLOSER 2018 (2018)
6. Chen, Y., Huang, J., Lin, C., Shen, X.: Multi-objective service composition with QoS dependencies. IEEE Trans. Cloud Comput. 7(2), 537–552 (2016). https://doi.org/10.1109/TCC.2016.2607750. http://ieeexplore.ieee.org/document/7563862/
7. Ding, S., Wang, Z., Wu, D., Olson, D.L.: Utilizing customer satisfaction in ranking prediction for personalized cloud service selection. Decis. Support Syst. 93, 1–10 (2017). https://doi.org/10.1016/j.dss.2016.09.001
8. Hayyolalam, V., Kazem, A.A.P.: A systematic literature review on QoS-aware service composition and selection in cloud environment. J. Netw. Comput. Appl. 110, 52–74 (2018). https://doi.org/10.1016/j.jnca.2018.03.003
9. Fischer, H.: A History of the Central Limit Theorem: From Classical to Modern Probability Theory. Sources and Studies in the History of Mathematics and Physical Sciences. Springer, New York (2011). https://doi.org/10.1007/978-0-387-87857-7
10. Hongzhen, X., Limin, L., Dehua, X., Yanqin, L.: Evolution of service composition based on QoS under the cloud computing environment. In: Proceedings of ICOACS 2016, pp. 66–69 (2016)
11. Jatoth, C., Gangadharan, G.R., Fiore, U., Buyya, R.: SELCLOUD: a hybrid multi-criteria decision-making model for selection of cloud services. Soft. Comput. 23(13), 4701–4715 (2018). https://doi.org/10.1007/s00500-018-3120-2
12. Jian, L., Youling, C., Long, W., Lidan, Z., Yufei, N.: An approach for service composition optimisation considering service correlation via a parallel max-min ant system based on the case library. Int. J. Comput. Integr. Manuf. 31(12), 1174–1188 (2018). https://doi.org/10.1080/0951192X.2018.1529435
13. Liu, Z.Z., Chu, D.H., Song, C., Xue, X., Lu, B.Y.: Social learning optimization (SLO) algorithm paradigm and its application in QoS-aware cloud service composition. Inf. Sci. 326, 315–333 (2016). https://doi.org/10.1016/j.ins.2015.08.004
14. Mezni, H., Sellami, M.: Multi-cloud service composition using formal concept analysis. J. Syst. Softw. 134, 138–152 (2017). https://doi.org/10.1016/j.jss.2017.08.016
15. Panda, S.K., Pande, S.K., Das, S.: Task partitioning scheduling algorithms for heterogeneous multi-cloud environment. Arab. J. Sci. Eng. 43(2), 913–933 (2017). https://doi.org/10.1007/s13369-017-2798-2
16. Seghir, F., Khababa, A.: A hybrid approach using genetic and fruit fly optimization algorithms for QoS-aware cloud service composition. J. Intell. Manuf. 29(8), 1773–1792 (2016). https://doi.org/10.1007/s10845-016-1215-0
17. Sousa, G., Rudametkin, W., Duchien, L.: Automated setup of multi-cloud environments for microservices-based applications. In: 9th IEEE International Conference on Cloud Computing (2016). https://doi.org/10.1109/CLOUD.2016.49
18. Thomas, M.V., Chandrasekaran, K.: Dynamic partner selection in Cloud Federation for ensuring the quality of service for cloud consumers. Int. J. Model. Simul. Sci. Comput. 08(03), 1750036 (2017). https://doi.org/10.1142/S1793962317500362. http://www.worldscientific.com/doi/abs/10.1142/S1793962317500362
19. Yimin, Z., Guojun, S., Xiaoguang, Y.: Cloud service selection optimization method based on parallel discrete particle swarm optimization. In: Proceedings of the 30th Chinese Control and Decision Conference, CCDC 2018, pp. 2103–2107 (2018). https://doi.org/10.1109/CCDC.2018.8407473
20. Zhou, J., Yao, X.: Multi-population parallel self-adaptive differential artificial bee colony algorithm with application in large-scale service composition for cloud manufacturing. Appl. Soft Comput. J. 56, 379–397 (2017). https://doi.org/10.1016/j.asoc.2017.03.017