Optimizing Services Selection in a Cloud Multiplatform Scenario

Everton Cavalcante, Thais Batista, Frederico Lopes
UFRN – Federal University of Rio Grande do Norte
Natal, Brazil
{evertonranielly, thaisbatista, fred.lopes}@gmail.com

Flavia C. Delicato, Paulo F. Pires
UFRJ – Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
{fdelicato, paulo.f.pires}@gmail.com

Noemi Rodriguez, Ana Lúcia de Moura
PUC-Rio – Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
{noemi, amoura}@inf.puc-rio.br

Reginaldo Mendes
SERPRO – Federal Service of Data Processing
Brasilia, Brazil
jrsmjr@gmail.com

Abstract—Services selection is an important challenge for applications that use a composition of services provided by different cloud platforms. This paper presents an optimized cloud services selection approach that evaluates each alternative set of services that composed an execution plan. This approach considers cost and quality parameters for each cloud service in the execution plan, and excludes coincident services in the calculations. Coincident services are those services present in all available execution plans and therefore equally contribute in the calculations encompassing costs and quality parameters regarding these plans. By excluding coincident services, the service selection process by itself should be performed more quickly since it involves a smaller number of services, but with the trade-off of running an additional algorithm for identifying the coincident services among the available ones. In order to evaluate such trade-off and illustrate the proposed approach, we present a case study and an experimental evaluation that compares our new approach with a previous one that considers all services that compose all available execution plans.

Keywords—Cloud Computing; semantic workflows; composition of semantic Web services; execution plans; cloud services selection.

I. INTRODUCTION

Cloud Computing is a new computing paradigm that enables ubiquitous, convenient, on-demand network access to a shared pool of configurable resources that can be rapidly provisioned and released with minimal management effort or interaction with the service provider [1]. With the advance of such paradigm, a single service offered by a cloud platform may not be enough to meet all the requirements of client applications. To fulfill such requirements, it may be necessary, instead of a single service, a service composition that aggregates services provided by different Cloud Computing platforms. However, current cloud platforms are not implemented using common standards, each one having its own APIs, development tools, and virtualization mechanisms, thus hampering the integration among services provided by different cloud platforms.

In order to mitigate such issues, our previous work introduced Cloud Integrator [2], a service-oriented middleware platform for composing, executing, and managing services provided by different Cloud Computing platforms. Cloud Integrator is based on SOA (Service-Oriented Architecture) and its implementation uses the Web services technology, thus exploring open standards and languages and enabling the integration of services provided by several service providers. According to the SOA approach, cloud platforms are service providers that provide services to cloud applications (clients), and Cloud Integrator works as a mediator between these players. The composition of services performed by Cloud Integrator is specified in terms of a semantic workflow that is composed of a sequence of abstract activities regarding the application, so that such activities must be performed by cloud services to achieve the application’s business goal.

Once the user abstractly specified a semantic workflow in terms of activities, Cloud Integrator performs a service composition based on this specification, i.e. searches for services that perform each activity specified in the workflow, and creates possible execution plans that contain a set of corresponding concrete Web services in an orchestrated way. Usually, there is more than one execution plan for a workflow and the number of these execution plans depends on the amount of available services with the same functionality in the environment in a given moment. Hence, it is necessary to perform a service selection algorithm that selects an execution plan to be executed among the available ones. To make this choice, the algorithm uses metadata about the services, such as Quality of Service (QoS) parameters and cost.

In our previous work [2] we presented a cloud services selection algorithm that evaluates the utility of each alternative execution plan based on calculations that encompass cost and quality parameters for each cloud service included in the execution plan. However, since such algorithm considers all services that compose all available execution plans, if an application requires a large number of services (as in the case of scientific computing applications, for example), the selection process becomes somewhat impracticable due to the increasing of the time spent by the service selection algorithm. Since the performance of the service selection algorithm can have great influence on the overall performance of the system [16], it is necessary to provide means to optimize the service selection process.
In this perspective, this paper presents a new approach for selecting cloud services that optimizes our former approach to evaluate alternative execution plans by not considering coincident services in the calculations. These coincident services are services that are present in all available execution plans and therefore equally contribute in the calculations encompassing costs and quality parameters regarding these plans. Intuitively, by excluding coincident services, the service selection process by itself should be performed more quickly since it involves a smaller number of services, but with the trade-off of running an additional algorithm for identifying the coincident services among the available ones. In order to evaluate such trade off and illustrate the proposed approach, we use as running example a cloud-based flight booking application that is also used in an experimental study that evaluates our new approach by comparing it with the one presented in our previous work [2].

This paper is structured as follows. Section II presents a cloud-based flight booking application, the running example used throughout this paper (Section II.A), and basic concepts necessary to fully understand the composition and service selection processes performed by Cloud Integrator (Section II.B). Section III presents the new approach for cloud services selection. Section IV reports an experimental study that aims to evaluate the service composition process and the approaches for the service selection. Section V discusses related works. Section VI contains final remarks.

II. BACKGROUND

A. Running example: A cloud-based flight booking application

Consider a simplified cloud-based flight booking application operated by a travel agency that manages flights from several airlines. For booking a flight, the user searches for available flights based on criteria such as departure and arrival airport and dates. If there are available flights, the user selects the desired flights and the system generates a booking regarding the selected flights. Next, the payment via credit card is executed and it is generated and stored a payment receipt containing all data about the booking and the client if the payment is successfully performed. Finally, a confirmation message is sent to the user. Figure 1 depicts the execution flow of these activities.

![Execution flow of the activities in the running example.](image)

**Figure 1. Execution flow of the activities in the running example.**

B. Basic concepts

In Cloud Integrator, a semantic workflow is an abstract representation of a workflow described in terms of activities, so that a workflow defines the sequence in which these activities must be performed in order to meet the application’s business goal. In the semantic workflow, each activity is specified by a tuple <task, object> (e.g. <store, file>, <send, message>, etc.) in which a task represents an operation implemented by one or more services and an object can be used for describing inputs, outputs, preconditions and effects related to a task. Thus, the services that perform each activity can be dynamically found at runtime based only on the <task, object> tuples. The task and object concepts are represented in terms of ontologies [3], which provide a common vocabulary for representing a specific domain with a high degree of formal expressiveness. In addition, ontologies prevent different semantic interpretations of a same set of information (thus avoiding ambiguity) and allow sharing the documented knowledge with other ontologies in order to reuse and improve the defined vocabulary.

Cloud Integrator uses two modalities of ontologies for representing concepts in the Cloud Computing context. The first one is named domain ontology, which models specific concepts regarding a given class of cloud applications and is structured in terms of tasks and objects. Figure 2 shows a partial representation of the FlightBooking domain ontology for the e-commerce application (presented in Section II.A) that describes tasks and objects related to this kind of application.

![Partial representation of the FlightBooking domain ontology.](image)

**Figure 2. Partial representation of the FlightBooking domain ontology.**

The second modality of ontologies used by Cloud Integrator is named cloud ontology and specifically refers to the Cloud Computing context. This generic and extensible ontology was inspired in the cloud ontology proposed by Han and Sim [4] and aims to represent the services provided by cloud platforms and the relationships between them. Such services are classified as: (i) cloud resources, which refer to resources provided by cloud platforms (IaaS services), and; (ii) application services, which may be SaaS and/or PaaS cloud services or even other traditional (non-cloud) services. All ontologies used by Cloud Integrator are described using the OWL (Web Ontology Language) language [5] by using classes, instances (individuals), properties and relationships.

The services that perform the specified activities are described as semantic Web services [6] using the OWL-S (Web Ontology Language for Web Services) [7] ontology language, which enables to describe Web services in an unambiguous, machine-interpretable way and increasing the automation degree of the service composition. Moreover, inference machines can be used to automatically discover and compose services through their inputs, outputs, preconditions and effects. Therefore, the application developer does not need to directly choose the services to be used at development time, thus enabling a greater flexibility for promoting on-demand access to the services. When executing a workflow, Cloud Integrator makes
inferences considering the workflow specification and the semantic Web services, thus performing the service composition and identifying which services should be selected to meet the business goal of the workflow. To do this, Cloud Integrator uses the service composition algorithm presented by Mendes et al. [8] that composes semantic Web services from a workflow specification by considering the required inputs and preconditions and the produced outputs and effects regarding each activity.

In order to execute a semantic workflow, it is necessary to create at least one concrete specification for the workflow, i.e. an execution plan that contains a set of orchestrated concrete Web services. The execution plans are built through an on-the-fly process of service discovery and composition according to the semantic interface of the selected services and the semantic workflow specification. Once the user abstractly specifies a semantic workflow that represents the activities of his/her application, Cloud Integrator performs a service composition based on this specification, i.e. searches for services that perform each one of the activities specified in the workflow, and creates possible execution plans by composing these services. In Cloud Integrator, the execution plans of a workflow are represented by a directed acyclic graph (DAG) in which each intermediate node represents a specific service and the directed edges represent the sequence of execution of the services. Figure 3 illustrates a DAG for the running example with 4 possible execution plans composed of services that perform the respective workflow activities. In this DAG, A abstracts a relational database service (such as Amazon RDS) and contains operations for queries and updates over the database. B represents an application service regarding the flight booking system itself. C is an application service for credit card payment. D abstracts a service for storing files (such as Amazon S3) regarding the receipts generated by the application. Finally, E is an application service for sending messages to the users.

Since different services with similar functionality may be available in the environment at a given moment, more than one execution plan can be created for a workflow. Hence, it is necessary to perform a service selection algorithm that selects an execution plan to be executed among the available ones. To guide this choice, the selection algorithm uses metadata about the services, such as QoS parameters, cost, etc. After selecting an execution plan, Cloud Integrator is able to execute it by sequentially invoking the services that compose it.

III. A NEW APPROACH FOR CLOUD SERVICES SELECTION

In our previous work [2], we presented a cloud services selection algorithm that encompasses two basic operations, namely: (i) aggregation, which computes the global value of a given parameter considering an execution plan as a whole, and; (ii) normalization, which fits the parameters in the same range of values in order to enable a uniform measurement of the quality of the execution plans. However, these operations consider all services that compose all available execution plans. When there is a relatively large number of services, the selection process becomes somewhat impracticable in terms of combining such services and calculating all aggregated values for each one of the considered parameters for each execution plan and then choosing the best one, thus considerably increasing the time spent by the service selection algorithm. The service composition and selection based on quality parameters is a combinatorial problem that is difficult to solve in terms of computational time for finding optimal solutions, as reported in some works in the literature [9, 10, 11].

In this paper we propose a new approach that strives to optimize the selection process performed by Cloud Integrator by not considering all coincident services in the calculations. These coincident services are services that are present in all available execution plans and therefore equally contribute with the same cost and quality values in any execution plan. Hence, the service selection process would in theory be faster because it considers a smaller number of services, but with a trade-off to run an additional algorithm for identifying the coincident services among the available ones beforehand. After identifying the coincident services, the selection process itself begins. It is noteworthy that the original service selection algorithm presented in our previous work [2] has not undergone any modification, i.e. the algorithm for identifying coincident services is just performed in a step prior to the selection algorithm. After that, the selection process considers only the services that are not coincident instead of all services. Figure 4 presents the algorithm to identify coincident services to be disregarded in the calculations performed in the process of selecting execution plans. Given a list \( A \) of activities that compose the workflow, for each activity \( a \) of the workflow \( (a \in A) \), the services that perform this activity \( a \) are enumerated by executing the getServicesByActivity procedure (line 3). Thus, if there is only one service that performs this activity \( a \) (line 4), then such service necessarily is in all execution plans and therefore it is marked as a coincident service, and added to the set of coincident services \( CS \) to be returned (line 5).

**Input:** \( A \) – list of activities that compose the workflow  
**Output:** \( CS \) – set of coincident services

1: \( CS \leftarrow \emptyset \)  
2: for each activity \( a \in A \) do  
3: \( S \leftarrow \text{getServicesByActivity}(a) \)  
4: if \( |S| = 1 \) then  
5: \( CS \leftarrow CS \cup S \)  
6: end if  
7: end for

Figure 4. Algorithm for identifying coincident services.

After identifying the coincident services, the selection process itself is started, as summarized in the algorithm presented...
in Figure 5. This algorithm receives as input the list of generated execution plans \( EP \) that compose the workflow, the set of coincident services \( CS \), and a list of weights \( W \) assigned by the user to the parameters regarding his/her preferences. The coincident services in the set \( CS \) are not considered in the operations performed at this stage of the selection process.

**Input:** \( EP \) – list of execution plans

**Output:** \( x \in EP \) – selected execution plan (with maximal utility)

1. **for each** execution plan \( p \in EP \) 
2. calculateAggregatedCost\( (p, CS) \)
3. normalizeAggregatedCost\( (p, PE) \)
4. end for

5. **for each** execution plan \( p \in EP \)
6. **for each** quality parameter \( q \)
7. calculateGlobalQuality\( (p, q, CS) \)
8. normalizeGlobalQuality\( (p, q, EP) \)
9. end for

10. calculateUtility\( (p, W) \)
11. end for

Figure 5. Execution plan selection algorithm.

The algorithm depicted in Figure 5 begins with the calculation of the (monetary) cost of each execution plan \( p \in EP \) (line 2) through a simple sum of the cost of each service that composes the current execution plan. Afterwards, this aggregated value is normalized (line 3) in order to fit it in the range between 0 and 1 by using the Equation 1:

\[
c_N = \frac{c_{\text{max}} - c_i}{c_{\text{max}} - c_{\text{min}}}, \quad c_{\text{max}} \neq c_{\text{min}} \quad (1)
\]

where \( c_N \) is the normalized cost and \( c_{\text{max}} \) and \( c_{\text{min}} \) are the greatest and the smallest aggregated values among the execution plans, respectively (if \( c_{\text{max}} = c_{\text{min}} \) then \( c_N = 1 \)). As pointed out in our previous work [2], the cost and the quality parameters are separately considered (in this order) because selecting an execution plan with the smallest cost does not necessarily imply that it has the best quality.

In the next step, the algorithm calculates the quality of each execution plan considering QoS parameters regarding the services that compose this execution plan. First, the global value of each quality parameter \( q \) for each execution plan \( p \in EP \) (line 7) is computed through specific aggregation functions for each parameter [10, 12]. Next, this value is normalized (line 8) in order to fit in the range between 0 and 1 [11]. Depending on their nature, some parameters are positive, i.e. the quality is better if the value is greater (e.g. the availability parameter), and other parameters are negative, i.e. the quality is better if the value is smaller (e.g. the response time parameter). Equations 2 and 3 respectively present the formulae for normalizing positive and negative parameters. In these equations, \( q_{\text{Ni}} \) is the normalized value of the parameter \( i, q_i \) is the global value of this parameter for the current execution plan, and \( q_{\text{max}}, q_{\text{min}}, q_{\text{max}} \) are the greatest and the smallest values of this parameter for all considered execution plans, respectively (if \( q_{\text{max}} = q_{\text{min}} \) then \( q_{\text{Ni}} = 1 \)). In this process, \( q_{\text{Ni}} \) results in a value between 0 and 1.

\[
q_{\text{Ni}} = \begin{cases} 
\frac{q_i - q_{\text{min}}}{q_{\text{max}} - q_{\text{min}}}, & q_{\text{max}} \neq q_{\text{min}} \\
0, & q_{\text{max}} = q_{\text{min}}
\end{cases} \quad (2)
\]

\[
q_{\text{Ni}} = \begin{cases} 
\frac{q_{\text{max}} - q_i}{q_{\text{max}} - q_{\text{min}}}, & q_{\text{max}} \neq q_{\text{min}} \\
1, & q_{\text{max}} = q_{\text{min}}
\end{cases} \quad (3)
\]

Finally, the utility of each execution plan \( p \in EP \) (line 10) is calculated. This utility is calculated as the weighted sum (Equation 4) between the normalized values of the parameters and the respective weights \( w \in W \) assigned by the user to these parameters. Thus, at the end of the selection process, the execution plan with maximal utility is selected. If there is more than one execution plan with maximal utility, one of them is randomly selected. In Equation 4, \( q_{\text{Ni}} \) and \( c_N \) are the normalized values for the quality parameter \( i \) and for the cost considering the execution plan \( p \), and \( w_i, w_e \in W \) are the weights assigned by the user to these parameters, so that \( w_i, w_e \in [0, 1] \) and the sum of all weights is equal to 1.

\[
u(p) = \sum_{i=1}^{w} q_{\text{Ni}}^{*} w_i + (c_N^{*} w_e) \quad (4)
\]

IV. Evaluation

The presented evaluation was performed under the following aspects: (i) the semantic composition process, and; (ii) the selection of execution plans by comparing the presented approach for cloud services selection with the one proposed in our previous work [2]. This experimental study used the running example presented in Section II.A and was performed in Linux Ubuntu 12.04 64-bits operating system by using a computer with an Intel® Core™ i7-3612QM 2.10GHz processor and 8 GB of RAM. Once the workflow was specified, 20 runs of the composition and selection processes were performed. For this evaluation, we used the Amazon RDS relational database service and the Amazon S3 storage service for implementing the application functionalities (as depicted in Section II.B) and created replicas of some services in order to have a variation in the number of possible execution plans created by different combinations of the services, thus resulting in 2, 4, 8, 12 and 18 execution plans. In addition, we have considered the availability and response time QoS parameters and generated fictitious values regarding the quality metadata for the services. More details of the experiment setup are available at http://consiste.dimap.ufrn.br/submissions/latincloud2012/.

A. Service composition

As previously mentioned, in the service composition process the workflow abstract specification in terms of activities is transformed into concrete service compositions, namely the execution plans. The services that perform each activity that composes the workflow in terms of its inputs, outputs, preconditions and effects are identified to be composed in order to generate the execution plans according to the activities sequence defined in the workflow specification. This process is computationally more expensive because it encompasses the analysis of the ontologies regarding each available service and the processing of XML documents (which is mandatory due to the use of Web services and Semantic Web technologies).
Table 1 presents the minimum (Min), maximum (Max), average (Avg) and standard deviation (StDev) values for the time spent by the service composition process considering different configurations of possible execution plans. In the simplest configuration (2 execution plans) the average time spent for the composition process was 502 ms, whereas the average time spent in the most complex configuration (18 execution plans) was 563 ms. As shown in Table 1, despite the increased number of possible execution plans due to the number of available services to perform the workflow activities, the time spent by the composition process does not increase significantly.

<table>
<thead>
<tr>
<th>Execution plans</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>StDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>474</td>
<td>575</td>
<td>502</td>
<td>28.051</td>
</tr>
<tr>
<td>4</td>
<td>493</td>
<td>578</td>
<td>524</td>
<td>28.688</td>
</tr>
<tr>
<td>8</td>
<td>507</td>
<td>590</td>
<td>538</td>
<td>28.546</td>
</tr>
<tr>
<td>12</td>
<td>535</td>
<td>614</td>
<td>563</td>
<td>25.554</td>
</tr>
<tr>
<td>18</td>
<td>566</td>
<td>645</td>
<td>597</td>
<td>26.023</td>
</tr>
</tbody>
</table>

### B. Selection of execution plans

Figure 6 presents the average time spent (in milliseconds) by the approach that considers all services and the approach that disregards coincident services in the service selection process. As shown in Figure 6, the time spent by both approaches is very small, but the approach that disregards coincident services present smaller average times than the approach that considers all services. In the simplest configuration (2 execution plans), the average time spent by the former selection approach was 0.385 ms against 0.335 ms for the new selection approach (a reduction of 12.99%). In the most complex configuration (18 execution plans), the average time spent by the former selection approach was 3.675 ms against 3.131 ms for the new selection approach (a reduction of 14.82%), thus confirming the hypothesis in which the new selection approach is more efficient in terms of performance and reduces the time spent by the selection process itself.

![Figure 6. Comparative chart for the average time spent (in milliseconds) by the selection service approaches.](image)

As it can be observed in Figure 6, the differences between the average times spent by the compared approaches are not greater than 1 ms. In this perspective, we have applied the Mann-Whitney nonparametric test [13] in order to verify the statistical significance of the obtained results. By adopting a significance level of 0.05 (5%), p-values that are smaller than 0.05 indicate that the approach that considers all services is significantly better than the approach that disregards coincident services, whereas p-values that are greater than 0.95 indicate that the approach that disregards coincident services is better than the approach that considers all services, and values in the range [0.05; 0.95] indicate that it is not possible to draw conclusions about significant differences between the approaches. As shown in Table 2, for all considered configurations of execution plans, the p-values obtained by the Mann-Whitney test are greater than 0.95, thus confirming that the approach that disregards coincident services in the selection process is significantly better than the approach that consider all services.

### Table 2. P-values resulted from the Mann-Whitney nonparametric statistical test comparing the service selection approaches.

<table>
<thead>
<tr>
<th>Execution plans</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-values</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.99996</td>
<td>0.99987</td>
<td>0.99875</td>
</tr>
</tbody>
</table>

A very important issue related to the use of the approach that disregards coincident services regards to the fact that it is necessary to run an additional algorithm for identifying the coincident services among the available ones. In this perspective, Table 3 presents the minimum (Min), maximum (Max), average (Avg) and standard deviation (StDev) values (in milliseconds) for the algorithm for identifying coincident services considering different execution plan configurations. Besides the average times are small, it can be observed that such algorithm has a good trade-off between the number of available execution plans to be analyzed and the number of coincident services among them, thus maintaining a relatively constant behavior between increasing the number of coincident services by reducing the number of execution plans, and the reducing the number of coincident services by increasing the number of execution plans. Moreover, the algorithm for identifying coincident services needs to be run only once (after the composition process) even in cases when the quality metadata regarding the services change over time and consequently it could be necessary to recalculate the quality of the execution plans in order to select another alternative execution plan that has a better quality than the current one.

### Table 3. Minimum, maximum, average and standard deviation values (in milliseconds) regarding the algorithm for identifying coincident services.

<table>
<thead>
<tr>
<th>Execution plans</th>
<th>Coincident services</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>StDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>15.200</td>
<td>28.195</td>
<td>18.541</td>
<td>3.071</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>13.710</td>
<td>35.080</td>
<td>17.848</td>
<td>0.515</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>14.801</td>
<td>22.103</td>
<td>17.504</td>
<td>2.134</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>13.991</td>
<td>28.798</td>
<td>17.499</td>
<td>3.314</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>13.279</td>
<td>20.425</td>
<td>15.919</td>
<td>1.775</td>
</tr>
</tbody>
</table>

### V. RELATED WORK

Service selection is an issue that has been deeply studied in the context of Web services mainly considering QoS parameters in such selection [14]. Since service-oriented computing is considered as the underlying conception of Cloud Computing, a natural way of thinking would be to extend or take advantage of traditional service selection approaches in order to...
address cloud services, of course considering the specificities introduced by this new paradigm. However, as we present in this section, to the best of our knowledge, the literature does not present many proposals for applying service selection techniques to the Cloud Computing context. Here we briefly discuss some of these works and highlight some interesting ideas provided by them and that could be applied to our approach in future works.

Wang et al. [15] propose a QoS-aware service selection approach to be applied in cloud environments. The authors use an approach similar to the one adopted in our work for calculating a QoS-based utility of generated service compositions (execution plans) and then selecting one of them. In addition, the approach employs a mixed integer mathematical programming technique for considering constraints over the composition and selection processes. However, in the approach rationale, it is necessary to know the internal transactions of the services and not always this kind of information is available when considering cloud services. In the strategy adopted by Cloud Integrator, the cloud services to be included in the execution plans (composition process) and to be selected (selection process) are handled in a higher abstraction level under the perspective of semantic Web services. In contrast to our simple approach, the adoption of a mixed integer mathematical programming technique could make the problem less understandable and an increase in the number of integer variables makes such model somewhat unfeasible in some practical scenarios since it would require a high computational time.

Finally, Kouki et al. [16] propose an approach to address the cloud service selection by considering QoS parameters and service-level agreements (SLAs), which are used by the user to specify constraints over the QoS parameters to be considered in the service selection process, where such constraints are mathematically modeled in terms of constraint programming. The authors also use an approach very similar to the one adopted in our work for calculating aggregated and normalized values for quality metadata regarding the services, so that the user can specify weights to the QoS parameters in order to select the most suitable services according to their preferences. However, our approach does not consider SLAs in the service selection process yet. We intend to consider this issue in future works.

VI. FINAL REMARKS

Cloud computing is an Internet-based computational paradigm where resources are provided as scalable, resilient, and on-demand services. This new paradigm and a realization of SOA pattern and Web services are widely employed for building distributed cloud applications. With the increasing number and diversity of services currently available in the Internet, selecting the most suitable service for a client application from a set of functionally equivalent candidates becomes an important research problem that poses several challenges. In our former work, we proposed a cloud services selection algorithm that evaluated the utility of each alternative set of services taking into account cost and quality parameters. However, such algorithm performed poor in situations where the application required a large number of services, e.g. in scientific computing applications. Therefore, we specified a new, enhanced version of our algorithm aimed at overcoming such limitation. In this paper we presented the new approach for selecting cloud services and the results of the performed evaluation and comparison with our previous work. We achieved promising results in which the new approach performed well even for an increasing number of services with variable behavior regarding the delivered quality of service.

ACKNOWLEDGMENTS

This work was partially supported by RNP through the AltoStratus CTIC Project and by CNPq through the CoSeNu Project. Flavia C. Delicato and Paulo F. Pires are also partially supported by FAPERJ under grants E-26/102.961/2012 and E-26/110.504/2012, respectively.

REFERENCES