World Applied Sciences Journal 28 (4): 463-472, 2013
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2013.28.04.1468

Performance Analysis of QoS-Based Web Service Selection Through Integer Programming

[1]*Seyed Hossein Siadat,* [2]*Alexandre Mello Ferreira,*
[3]*Tala Talaei Khoei and* [4]*Ahmad Reza Ghapanchi*

[1]IT-Management Group, Shahid Beheshti University, Iran
[2]Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milan, 20133, Italy
[3]Mashahd Islamic Azad University, Iran
[4]School of Computer Engineering, Isfahan University, Iran

Abstract: With the rapid propagation of Web Services and the increase of functionally similar Web Services, the issue of selecting them based on their quality attributes is becoming very popular among the research community and practitioners. Quality of Services (QoS) are distinguishing factors for users in selecting Web Services, when there are multiple Web Services with overlapping or equal functionality. Therefore, a choice needs to be defined in order to identify which services will be participating in the Web Service composition to perform a sequence of tasks. This work presents a quality model and a selecting method from a quality perspective that maximizes the user satisfaction. We use an Integer Programming (IP) technique as a selection approach for allocating services to tasks. We implement different variants of the selection process and test execution times for finding the optimum solution for the service selection problem, to capture all the needs of real scenarios. In particular, we vary a number of parameters namely number of available services, number of tasks in the service composition and probability for a service to match requirements of a task. The results we obtain show the benefit of IP when applied to the service selection problem.

Key words: Web service selection · Quality of service · Quality model · Web service composition · Integer programming

INTRODUCTION

The adoption of software-as-a-service model (SaaS) has exponentially increased in the last years, widespread in a variety of application domains leading to costs reduction and quality increases. In fact, such model is one of the bases of cloud computing paradigm providing on-demand service applications over a service oriented, dynamic resource provisioning and utility-based models [1]. Essentially, a service application is made up by compositions of web services which perform a sequence of tasks and are, basically, distributed applications that are *published, discovered* and *used* through the Web using standard protocols.

With the rapid propagation of web services available and the increase of functionally similar web services, the issue of *selecting* them based on their quality attributes is becoming very popular among the research community and practitioners. Quality of Services (QoS) are distinguishing factors for users in selecting web services, when there are multiple web services with overlapping or equal functionality. Therefore, a choice needs to be defined in order to identify which services will be participating in the Web Service composition considering functional and non-function attributes combined.

Typical non-functional properties include availability, throughput, cost and response time and they are often referred to as *quality dimensions*. First of all, in order to reason about QoS properties, there is a need for a quality

Corresponding Author: Ahmad Reza Ghapanchi, School of Computer Engineering, Isfahan University, Isfahan, Iran.

model to define the description of quality from both provider and requester sides. Second, mechanisms are required for selecting the candidate services and `eventually the best service for a specific task from a quality perspective. In the area of web service composition, QoS-based web service discovery is the process of *matchmaking* and *selecting* functionally equivalent WSs in a composition fashion [2].

The *matchmaking* phase aims to find out functional equivalent services through a matching process based on the user functional needs and functional capabilities of available services in the system's registry for each sequential task. The result is a group of functional-equivalent candidate web services appropriate for a specific task, which may differ in their non-functional characteristics. The functional matchmaking is a well-known problem which is successful solved by [3] and it is not the focus of this work as better explained in the composition model.

The *selection* phase consists in ranking and choosing the best service for each task according to global and local constraints defined by the user. In more details, the problem consists in selecting the best available service at runtime among all candidate ones for each task in a composition such that the choice maximizes user satisfaction over maximum and minimum quality attributes and, at the same time, do not violate either global constraints or structure limitations [4].

This work presents a quality model and selecting method from a quality perspective that eventually maximizes the user satisfaction. We use an Integer Programming (IP) technique as a selection approach for allocating services to tasks. The remainder of the paper is organized as follows. We start by briefly reviewing related work in Section 2 and afterwards define a web service composition model and quality model for both elementary and composite services in Section 3. In Section 4, we introduce our integer programming approach for the web service selection problem. The implementation using IP tools is explained in Section 5. Section 6 presents the evaluation and result and finally Section 7 concludes the paper.

Related Works: Web service selection is one of the fundamental parts in service oriented architecture both from functional and non-functional perspective. There is much previous work on service selection techniques policies based on, for example, reputation system [5, 6], semantic annotations [7] and multi attribute utility theory [8]. The system built to use these policies is often uses a Service Level Agreement (SLA) language to describe their services and contracts, such as [9, 10]. Dynamic service selection based on user requirement is also presented in literature [11]. However, most of them consider services individually and do not pay a good attention in selecting Web Services in a composite environment. Besides, Quality of Service (QoS) aspects mostly have not been considered.

Since many web services can provide a same functionality for a given task, therefore selecting web services from a QoS perspective could be a distinguishing factor which is getting a huge interest in the literature. A quality model is required for performing the service selection mechanism between service provider and requester. With this regards, a survey on service quality description is represented in [12]. Different techniques have been used to solve the problem of QoS-based web service selection in literature, such as ontology-based [13], constraint programming [14] and integer programming [2, 15]. Each of which has its own advantages and disadvantages is only applicable in certain circumstances. For example, Zeng *et al*. [2] uses mixed integer programming to calculate optimal web service execution plan in compositions when global user constraints are required. In a similar approach, Ardagna *et al*. [16] uses the linear programming model to include local constraints. However, one main problem of these approaches and in general linear programming techniques is that they are very useful when the size of problem is small, whereas they have limited scalability due to the exponential time complexity.

Therefore, in our approach, we investigate performance analysis of IP based web service selection. While the objective is to maximize the overall quality, we analyze implementation of different selection methods and their execution times comparison. In particular, we compare four different implementations of the service selection problem using different synthetic, yet realistic, workloads. We show how the performance changes when changing objective function and set of constraints. With this regards, we examine the impact of some parameters on the processing time, namely the number of services, the number of tasks and the matching probability.

Web Service Composition Model: We now introduce the *Web Service Composition Model*, which defines the properties of available WSs and the relationships between WSs and tasks. As already mentioned, such relationships are discovered during the *matchmaking* process and represent the input data for the *selection* process we address.

In the Web Service Composition Model we have a service composition (the *execution plan*) with a set of m tasks. Let us call T the set of all tasks $t_j, j \in [1..m]$. We then have a set S of n available services, $s_i, i \in [1..n]$. The matchmaking process binds available services and tasks by specifying the subset of candidate services that can be considered good for each task. A service s_i is good for a task t_j if it provides all the functionalities required by t_j. To express this relation we introduce a *dependency* parameter, d_i^j defined as follows:

$$d_i^j = \begin{cases} 1, if\ service\ i\ is\ good\ /\ candiadte\ to\ task\ j \\ 0, otherwise \end{cases} \quad (1)$$

Moreover, each service can safely execute only a limited number of tasks. This number is not fixed, but depends from the particular service. We model this aspect by defining a parameter c_i for each service s_i, representing the maximum capacity (in terms of number of tasks) of the service. In other words, each service s_i can be allocated to c_i tasks, at most.

Web Service Quality Model: Since the web service selection process is based on non-functional properties of services, we need to define a *Web Service Quality Model* for both elementary services and composite services. We assume that a set of h qualities (or properties) is defined for each service. Properties may include (a) *price*, which is the fee/cost that a service requester has to pay to the service provider for the service invocation, (b) *execution time*, which is the expected duration in time that a service spends to fulfill a service request and so forth. Accordingly, we define a parameter, q_i^k $k \in [1..h]$ that specifies the value of the quality k for service i. This way we are defining a quality matrix Q, specifying a value for each quality in each service. The quality matrix needs to be scaled since there are quality metrics having different types. Using a *Simple Additive Weighting* (*SAW*) the matrix Q will be scaled such that:

$$v_i^k = \begin{cases} \dfrac{q_{\max}^k - q_i^k}{q_{\max}^k - q_{\min}^k}, q_{\max}^k - q_{\min}^k \neq 0 \\ 1, \text{otherwise} \end{cases} \quad (2)$$

$$v_i^k = \begin{cases} \dfrac{q_i^k - q_{\min}^k}{q_{\max}^k - q_{\min}^k}, q_{\max}^k - q_{\min}^k \neq 0 \\ 1, \text{otherwise} \end{cases} \quad (3)$$

where is the minimum value for the quality k among all available services, while is the maximum value for the quality k among all available services. Note that Eq. (2) works with for *negative* QoS properties (the higher value the lower quality). An example of this type of quality could be the price of the web service. Instead, Eq. (3) works for *positive* parameters (the higher value the higher quality). An example of this type of quality could be the availability of the web service.

After this scaling phase, we are able to define an overall quality value/score for each web service. The overall score depends from the values v_i^k and from a weight w^k assigned to each quality. The weight reflects the relevance of the quality from the user's perspective. Accordingly, we define the score of each service as follows.

$$qVal_i = \sum_{k=1..k} v_i^k w^k \ \therefore \forall i \in S \quad (4)$$

The weights need to satisfy the following properties, defining a convex combination of w coefficients:

$$\sum_{k=1..h} w^k = 1$$
$$0 \leq w^k \leq 1 \quad (5)$$

As a final remark, we observe that the price of a service usually has an important role in the selection process. Indeed, sometimes there exists a maximum budget that we can pay for the composition, which is fixed and independent from the other quality dimensions.

For this reason, not only we embed the cost of a service inside the *qVal* parameter, but we also reserve a set of parameters f_i, representing the price of each service. These parameters will be used to generate constraints on the maximum budget for the composition, when required.

Problem Formulation: Defining a quality model, we are proposing an integer programming approach for selecting the best candidate service among available ones. The selecting criterion is the maximization of the total quality of a service while satisfying the capacity constraints defined for each service. In this Section, we provide a precise formulation of the problem in terms of IP. We make use of the parameters defined so far, while explicitly introducing decision variables, objective function and constraints. In the web service selection problem we defined, we need to assign a service to each task. Accordingly we introduce a set of variables x_i^j defined as follows:

$$x_i^j = \begin{cases} 1, \text{ if service } i \text{ is assigned to task } j \\ 0, \text{ otherwise} \end{cases} \quad (6)$$

We then introduce the following constraints. First, we need to select only a single service for each task.

$$\sum_{i=1..n} x_i^j = 1 : \forall j \in T \quad (7)$$

We then need a set of constraints to specify that a service can be assigned to a task only if it is good for the task.

$$x_i^j \le d_i^j : \forall i \in S, \forall j \in T \quad (8)$$

Additionally, we must ensure that the number of tasks allocated to a service does not exceed the capacity of the service.

$$\sum_{j=1..m} x_i^j \le c_i : \forall i \in S \quad (9)$$

Finally, if we have a limited maximum budget for the composition, says B, we need to include a budget constraint.

$$\sum_{i=1..n} f_i \sum_{j=1..m} x_i^j \le B \quad (10)$$

The objective function tries to maximize the overall quality of the selection process and is defined as follows:

$$\max \sum_{i=1..n} qVal_i \sum_{j=1..m} x_i^j \quad (11)$$

Notice that the given objective function selects services for each task by only looking at the overall quality of the assignment. While this is reasonable for many scenarios, still it does not offer any guarantee on the quality reserved to single tasks, which could be useful in particular applications.

Therefore, we may provide an alternative definition of the objective function that aims to maximize the minimum quality associated to a single task in the service composition. To do so, we need to introduce an additional variable, say K, with the following constraint.

$$K \le \sum_{i=1..n} qVal_i x_i^j : \forall j \in T \quad (12)$$

Given these premises, the objective function becomes:

Implementation: In this section we describe how the Web Service Selection problem has been implemented using IP tools. Since we wanted to test the overall time needed to find the optimum selection of services under different conditions, we provide four different implementations:

Quality Maximization Problem: the goal is to maximize the overall service quality, while satisfying the capacity constraints.

Minimum Quality Maximization Problem: the goal is to maximize the minimum quality for a single task, while satisfying the capacity constraints.

Quality Maximization Problem with Budget: like the Quality Maximization Problem, but we also have a maximum budget.

Minimum Quality Maximization Problem with Budget: like the Minimum Quality Maximization Problem, but we also have a maximum budget.

With these four cases, we cover all the needs coming from real applications. In particular, we can use formulations *i.* and *iii.* when the quality of the overall composition is the most important aspect in the selection process. This is probably the most frequent case, since the composite quality coming from all tasks directly influences the utility of the service from the users' perspective. Formulation *i.* does not include a budget constraint: this is reasonable in various scenarios, in which services have identical, or at least similar, costs and the overall budget for the composition is very high, so that we may be interested only in maximizing the quality. When this is not true, for example because services with a high quality have a very high cost, we can use formulation *iii.*

Even if less frequent, the case in which we are interested in maximizing the minimum quality reserved to a single task is still important. Some times there are critical tasks involved into the composition that need minimum quality guarantees. In this case, we want to be sure that single tasks do not pay too much in the maximization problem, so we may safely achieve a lower quality for the composition, to guarantee minimum values for single tasks. Also in this case we provide two different implementations, one without the budget constraint (*ii.*) and one with the additional budget constraint (*iv.*).

To describe and solve the problems mentioned, we used GLPK (GNU Linear Programming Kit) which is an open source and free set of libraries developed to solve mainly linear programming (LP) and mixed integer programming (MIP) [17]. GLPK can be called directly inside C/C++ programs, as a stand-alone solver. However, it also offers a program definition language, called GMPL, which is a subset of the well-known AMPL [18]. In the following, we make use of the GMPL syntax to describe the implementations of the Web Service Selection problem.

Testing Scenarios: In order to test the performance of our approach, we created a generator of synthetic workloads using Python [19]. This allowed us to test the execution time for finding the optimum solution for the Service Selection Problem, while varying a number of parameters. More in particular, we focused on the following parameter:

- Number of available services;
- Number of tasks in the service composition;
- Probability for a service to match the requirements of a task.

The number of available services and the number of tasks in the service composition greatly influence the dimension of the problem and more in particular the number of involved variables and constraints. Accordingly, we expect to experience better results with small numbers. It is worth mentioning that in realistic scenarios it is highly probable to have a small number of tasks involved in the composition [20], while it is possible for the number of available services to be significantly higher, especially in an open world, in which services belonging to the most disparate administrative domains can be combined together. As a consequence we are interested in studying how the problem scales when the number of services increases.

The matching probability is another key aspect in the problem: indeed, it determines the number of good services for each task. The higher is the matching probability, the larger is the number of feasible solutions for the problem. As we said, realistic scenarios involve a large number of heterogeneous services: as a consequence, the matching probability for single tasks is not very high. Generally speaking, we can say that it will never exceed a value of 10%. We tested our problem with three different values of matching probability: 1%, 5% and 10%. All other parameters involved in the problem pay a small role. For each of them we imposed fixed upper and lower bound and then let our workload generator select them randomly from a uniform distribution within the given bounds. Minimum and maximum bounds for parameters are shown below.

$$\begin{aligned} 1 \le qVal_i \le 10 : \forall i \in S \\ 1 \le c_i \le 10 : \forall i \in S \\ 1 \le f_i \le 10 : \forall i \in S \end{aligned} \quad (13)$$

For implementing (*ii.)* and (*iv.)* that make use of budget constraints, we computed the maximum budget B as follows:

$$B = 4m \quad (14)$$

In other words, the budget is equal to the number of tasks in the problem multiplied by 4, which is smaller than the average price of services (to guarantee that the constraints is actually used) but high enough to allow for feasible solutions.

Constrains: After discussing how the parameters of the testing scenarios have been generated, we now introduce the set of constraints as written in GMPL/GLPK. The first line guarantees that only one service is selected for each task. The second one states that a service is selected for a task only if it is good for it (using the d_i^j parameter, as described in the previous Section). The third line introduces a capacity constraint for each service, using the set of parameters c_i. Finally the last line introduces budget constraints: this constraint is used only for implementing (*ii*) and (*iv*.).

Algorithm 1: Problem constraints

1: singleServiceConstraint (j ∈ 1..m): $\sum_{i=1..n} x[i,j] = 1$;

2: selectGoodConstraint (i ∈ 1..n, j ∈ 1..m): $x[i,j] \le d[i,j]$;

3: capacityConstraint (i ∈ 1..n): $\sum_{j=1..m} x[i,j] \le c[i]$;

4: totalBudgetConstraint: $\sum_{i=1..n} f[i] * \sum_{j=1..m} x[i,j] \le B$;

Quality Maximization Problems: The implementation of the objective function for the Quality Maximization Problems (problems (*i*.) and (*iii.)* above) is translated into GMPL/GLPK as follows.

Algorithm 2: Quality maximization problem

1: maximizeTotalQuality: $\sum_{i=1..n} q[i] * \sum_{j=1..m} x[i,j]$;

Minimum Quality Maximization Problems: The implementation of the objective function for the Minimum Quality Maximization Problems (problems (*ii*.) and (*iv*.) above) is translated into GMPL/GLPK as follows.

Algorithm3: MaxMin problem

1: kDefinition: $K \geq \sum_{i=1..n} f[i] * \sum_{j=1..m} x[i,j]$;

2: maximizeMinQuality: K;

Notice in particular how the maximization of the minimum quality requires the introduction of a new constraint, which is used to define the variable k to be maximized.

Lagrangean Relaxation: Besides testing the problem with different implementations and different input parameters, we also tried to manually apply the Lagrangean Relaxation [21] to our problem, in order to solve it iteratively and compare the benefits of the method. In particular, we were interested in two aspects: on one hand we wanted to test the absolute speed of this algorithm; on the other hand, we wanted to understand if stopping the algorithm after a number of iterations could bring good approximations with a reduced computation time. The implementation of the Lagrangean Relaxation in GMPL/GLPK is shown below.

Algorithm4: Lagranean relaxiation

1: param p(1 @ 1..n) @ $q_{max} - q[i]$;

2: minimizeTotalPrize: $\left(\sum_{i=1..n} p[i] * \sum_{j=1..m} x[i,j]\right) - \lambda * \left(B - \sum_{i=1..n} f[i] * \sum_{j=1..m} x[i,j]\right)$;

In the first line we define a new parameter p_i for each service i @ S. This is defined as $p_i = q_{max} - q_i$, where q_{max} is the maximum value of quality among all available services. In other words, parameter p_i is the reverse of the quality parameter q_i: the lower is p_i the better is the selection of service i. We used this conversion from q to p to transform our problem into a minimization problem. Accordingly, the new objective function for our problem should become:

$$\sum_{i=1..n} p_i \sum_{j=1..m} x_i^j \quad (15)$$

By applying the lagrangean relaxation we relax a constraint (remove it) and use it as a negative weighted contribution in the objective function. The weight of the contribution is called λ and can be a single value, or a vector. In our case, we apply lagrangean relaxation to implementation (*iii*.) (which is the most common for real scenarios) and we relax the constraint on budget. As a consequence the new objective function becomes:

$$z = \left(\sum_{i=1..n} p_i \sum_{j=1..m} x_i^j\right) - \lambda\left(B - \sum_{i=1..n} f_i \sum_{j=1..m} x_i^j\right) \quad (16)$$

where the second part represents the contribution of the budget constraint weighted by λ. The translation of this objective function in the GMPL/GLPK language is shown in line two of the above algorithm. The idea in the lagrangean relaxation is to start solving the problem for a given value of λ. This problem should be easier, since it has fewer constraints. After finding a result, we can compute the derivative of z w.r.t. λ and see if we should decrease λ or increase it to move near its maximum value. Notice that in our case the λ parameter is a single number, so we can compute the derivative of z in a simple way, by looking at the objective function. In particular, the value of the derivative of z is:

$$\frac{dz}{d\lambda} = -\left(B - \sum_{i=1..n} f_i \sum_{j=1..m} x_i^j\right) \quad (17)$$

Since GMPL is less expressive than AMPL and does not allow expressing iterations, we implemented a script in Python to compute the value of the derivative of z at each iteration and to determine the new value of λ.

Evaluation: In this Section, we present the results we obtained using the implementations of the problem described in the previous section and our workload generator.

Methodology: To evaluate the performance of our implementation, we run each test multiple times, using different seeds to compute the random values. After each run, we computed the 95% confidence interval and we continued until the interval became smaller than 1% w.r.t. the value of the measure. This approach is used to guarantee that the results do not depend from the particular workload generated. All results are obtained using an Intel Core2Duo laptop running at 2.53GHz with 4GB of main memory.

RESULTS

We first discuss the results we obtained with the implementations of the Quality Maximization and Minimum Quality Maximization with budget constraints. They represent the most complete problems, involving both constraints on the capacity of services and constraints on the price of services. As said in the previous Section, we are mainly interested in studying how the execution time of our implementation changes while varying the number of available services. To test this aspect we fixed the number of tasks to 100. Notice that this is a very high number, which is highly improbable to be found in realistic scenarios. In a sense, we are using it since it represents a sort of upper bound for real applications.

Fig. 1 and Fig. 2 show the results we obtained when considering the Quality Maximization and the Minimum Quality Maximization problems, respectively. In each graph, we plot the execution time with a different number of available services. Notice that we also changed the probability of matching, using three values: 1%, 5% and 10%. By looking Fig. 1 and Fig. 2 some interesting aspects emerge. First of all, we notice that the execution time is very small; they never exceed 4 seconds. These values are particularly good if you consider that the quality of services changes rarely. Accordingly, we do not need to compute the selection process again and again, but we can safely use discovered values for long time to execute the task, at least until new services are discovered, or existing ones change their properties.

Second, we can observe that the time needed for the computation grows almost linearly with the number of services. This result is of primary importance: indeed, it allows our implementation to scale with the number of services. This is good, since the number of available services will probably continue to grow in the future. Third, we notice that the matching probability has only a marginal impact on the performance of the solver. In particular, increasing the matching probability from 1% to 10% increases the execution time by at most 1 second. Fourth, the performance of the Quality Maximization problem and those of the Minimum Quality Maximization problem are almost identical, with the second one performing only a little worse, probably because of the increased number of variables and constraints it needs to manage.

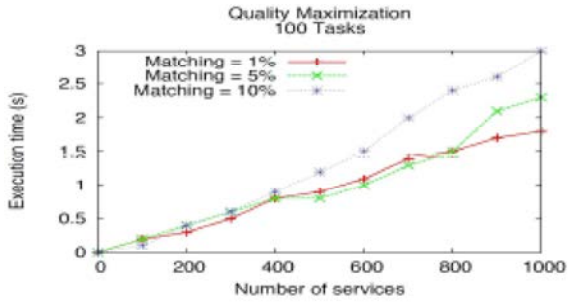

Fig. 1: Quality Maximization Problem

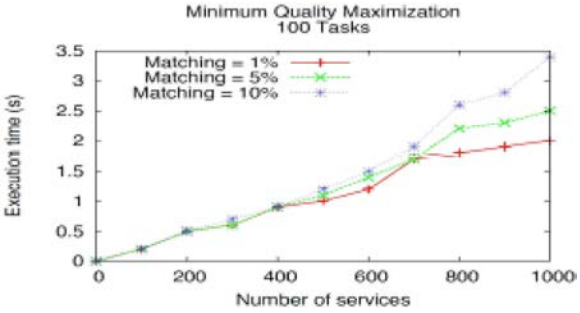

Fig. 2: Minimum Quality Maximization Problem

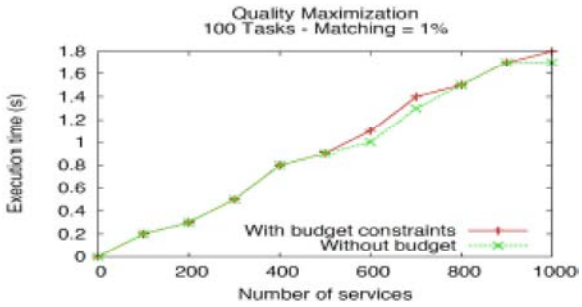

Fig. 3: Quality Maximization Problem - Cost Budge Constraint

Cost of the Budget Constraint: After presenting the performance we obtained with the Service Selection problem including the budget constraint, we remove it to study if it presents a high impact on the performance of the solver. Fig. 3 and Fig. 4 show the results we obtained when considering the Quality Maximization and the Minimum Quality Maximization problems, respectively.

By looking Fig. 3 and Fig. 4 we observe that the budget constraint does not influence the performance significantly. More in particular, in both cases, we observe a maximum decrease of execution time that is under 10%.

Worst Case: Increasing the Number of Tasks: While we are interested in using the IP implementation only for a limited number of tasks (as said, 100 tasks is an upper

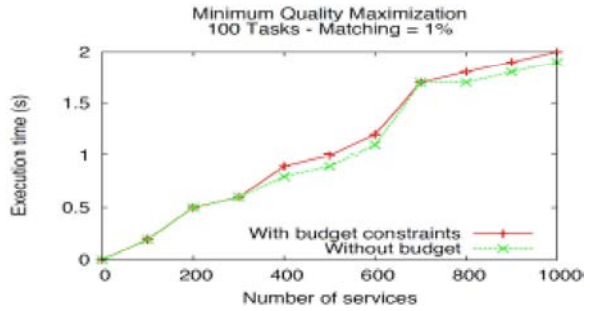

Fig. 4: Minimum Quality Maximization Problem - Cost of Budge Constraint

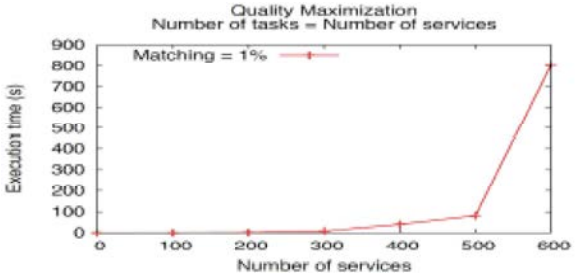

Fig. 5: Quality Maximization Problem - Increasing the number of Tasks

Fig. 6: Quality Maximization Problem - Lagrangean Relaxation

bound for realistic scenarios), we also want to test the limit of this approach. To do so, we changed the way workloads were generated and we did some experiments with a number of tasks constantly equal to the number of available service. Figure 5 shows the results we obtained with the Quality Maximization problem.

As we can easily see, in this case the computation cost increases significantly with the number of services. Indeed, by making the number of services and the number of tasks grow at the same time, we are increasing the number of constraints at a quadratic speed. However, what we experience here is an increase in the computation time that is exponential. This means that, in its general form, the Service Selection problem is not easy. However, it is solvable almost in real-time in all realistic scenarios.

Lagrangean Relaxation: Finally, we tested the performance of the implementation that makes use of the lagrangean relaxation. Fig. 6 shows the results we obtained. The graph shows the results with a maximum number of 25 iterations, which allow for a high precision in the detection of the maximum value of λ. In many executions, the minimum value is actually achieved, while in others the limitation on the number of iterations is reached before arriving to the minimum. In any case we reached the optimum value for the objective function, which means that we always arrived to an approximation on the value of λ that is sufficient to compute the correct value for the objective function.

In general, what we observe is that the execution time when using the lagrangean relaxation becomes much higher. Indeed, as we have seen before, the budget constraint that we are removing in the lagrangean relaxation does not make the problem much easier to solve. This means that, by applying lagrangean relaxation, we are only solving the problem without the budget constraint many times and we are paying the price of the iterations plus the additional cost of changing the value of λ and loading the problem with the new values in the objective function. This can easily explain the results we got.

On the other hand, we were interested in lagrangean relaxation to see if it was possible to find a good sub-optimal solution with a reduced execution time. This could be useful in scenarios involving frequent service reconfigurations: in these scenarios, it could be necessary to perform service selection repeatedly and consequently a good approximation of the optimum selection could be the right choice. However, also in this case, the lagrangean relaxation does not seem to be the right answer. Indeed, while we observe that, a good approximation (less than 1% of difference in the objective function) is reached in a limited number of iterations (usually less than 10); the processing time is still much higher than those of the normal execution are.

As a final remark, we noticed that most of the execution time was spent to find an integer solution, after an optimum solution had been found with the relaxation of integrality constraints. Accordingly, we can say that, if execution time has an important role, we can give an approximation (without offering guarantees on its distance from the optimum) by stopping the algorithm soon after a solution for the LP problem has been found and using some heuristics to move from the given solution to a feasible one with all integral variables.

CONCLUSION

In this paper we have presented a quality model for services and a linear programming approach to the service selection problem. The service selection problem consists in assigning services to a set of tasks according to their quality. The objective is to maximize the overall quality.

We compared four different implementations of the service selection problem using different synthetic, yet realistic, workloads. We showed how the performance, in terms of execution time needed to find the optimum solution, changed when changing the objective function and the set of constraints. We also studied the impact of some parameters on the processing time, namely the number of services, the number of tasks and the matching probability. Finally, we tried to apply a lagrangean relaxation methodology, to obtain good approximations with a reduced processing time. However, we observe how the methodology does not apply to the problem and we discussed possible reasons for that.

As a conclusion, we obtained interesting results, which are very good for almost all realistic scenarios. Indeed, with a number of tasks equal to 100, we could find the optimum assignment in few seconds (or event fractions of seconds). While we show how the problem does not scale in the most general case, this approach showed its benefits in all the scenarios needed by real applications.

ACKNOWLEDGEMENT

This work was partially supported by the Industrial Strategic technology development program (10040142, Development of a qualitative customer feedback analysis and evaluation method for effective performance management in B2C Industry) funded by the Ministry of Knowledge Economy (MKE, Korea).

REFERENCES

1. Zhang, Q., L. Cheng and R. Boutaba, 2010. Cloud computing: state-of-the-art and research challenges. Internet Services and Applications, 1: 7-18.
2. Zeng, L., B. Benatallah, M. Dumas, J. Kalagnanam and Q.Z. Sheng, 2003. Quality driven web services composition. Proceedings of the 12th International Conference on World Wide Web, pp: 411-421.
3. Plebani, P. and B. Pernici, 2009. URBE: Web Service Retrieval Based on Similarity Evaluation. IEEE Transactions on Knowledge and Data Engineering, 21(11): 1629-1642.
4. Ferreira, A.M., K. Kritikos and B. Pernici, 2009. Energy-aware design of service-based applications. Proceedings of the 7th International Joint Conference on Service-Oriented Computing, pp: 99-114.
5. Maximilien, E.M. and M.P. Singh, 2004. Toward Automatic Web Services Trust and Selection. Proceedings of the 2nd International Conference on Service Oriented Computing, pp: 212-221.
6. Vu, L.H., M. Hauswirth and R. Meersman, 2005. QoS-based Service Selection and Ranking with Trust and Reputation Management. On the Move to Meaningful Internet Systems: CoopIS, DOA and ODBASE, pp: 37-45.
7. Sirin, E., B. Parsia and J. Hendler, 2004. Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. IEEE Intelligent Systems, 19(4): 42-49.
8. Seo, Y.J., H.Y. Jeong and Y.J. Song, 2005. A Study on Web Services Selection Method Based on the Negotiation through Quality Broker: A MAUT based Approach. Proceedings of the first International Conference on Embedded Software and Systems, pp: 65-73.
9. Skene, J., D.D. Lamanna and W. Emmerich, 2004. Precise Service Level Agreements. Proceedings of 26th International Conference on Software Engineering, pp: 179-188.
10. Ludwig, H., A. Keller, A. Dan, R.P. King and R. Franck, 2003. Web Service Level Agreement (WSLA) Language Specification. Document WSLA-2003/01/28, IBM Corporation, pp: 1-110.
11. Casati, F. and M.C. Shan, 2001. Dynamic and adaptive composition of e-service. Information System, 26(3): 143-163.
12. Benbernou, S., I. Brandic, C. Cappiello, M. Carro, M. Comuzzi, A. Kertesz, K. Kritikos, M. Parkin and B. Pernici, 2010. Modeling and negotiating service quality. Service research challenges and solutions for the Future Internet, pp: 157-208.
13. Oldham, N., K. Verma, A. Sheth and F. Hakimpour, 2006. Semantic WS-agreement partner selection. Proceedings of the 15th International Conference on World Wide Web, pp: 697-706.
14. Cortes, A.R., O.M. Diaz, A.D. Toro and M. Toro, 2005. Improving the automatic procurement of web services using constraint programming. Cooperative Information Systems, 14(4): 439-468.
15. Kritikos, K. and D. Plexousakis, 2009. Mixed-integer programming for QoS-based web service matchmaking. IEEE Transaction on Service Computing, 2(2): 122-139.

16. Ardagna, D. and B. Pernici, 2007. Adaptive Service Composition in Flexible Processes. IEEE Transaction on Software Engineering, 33: 369-384.
17. GNU. 2001. GNU Linear Programming Kit Web Site. http://www.gnu.org/software/glpk/. Visited June 2011.
18. AMPL. 2011. AMPL Official Web Site. http://www.ampl.com/. Visited June 2011.
19. Python, 2011. Python Official Web Site. http://www.python.org/. Visited June 2011.
20. Papazoglou, M.P., P. Traverso, S. Dustdar and F. Leymann, 2003. Service-oriented computting. Communications of the ACM, 46: 25-28.
21. Geoffrion, A.M., 1974. Lagrangean relaxation for integer programming. Approaches to Integer Programing, 2: 82-114.