A Comparative Analysis of Multicriteria Approaches for Cloud Services Selection

Hivana Alice Medeiros de Macedo*
Departamento de Informática e Matemática Aplicada – Universidade Federal do Rio Grande do Norte
Caixa Postal 1524 – Campus Universitário – 59078-900 – Natal/RN – Brasil
hivanamacedo@lcc.ufrn.com

Carlos Eduardo da Silva, Luciano Ferreira
Escola de Ciência e Tecnologia – Universidade Federal do Rio Grande do Norte
Caixa Postal 1524 – Campus Universitário – 59078-900 – Natal/RN – Brasil
{carlos.silva,lucianoferreira}@ect.ufrn.br

ABSTRACT

With the visibility that cloud computing has been acquiring, it is observed that the provision of computational resources as services is growing increasingly, leading to a large number of services being offered. These services have heterogeneous managerial and technical specifications, so that organizations wishing to use such services need mechanisms to assist in the decision making process to choose one that best suits their needs. This paper addresses the issue of selection of cloud services based on the analysis and use of methods, models and algorithms for selection which meet multiple criteria. We present an unified approach for specifying user objectives, capturing properties of available services, and applying a selection method. Our approach is evaluated by combining two multi-criteria methods for selecting real cloud computing services.

KEYWORDS. Service Selection, Electre, Cloud Computing, Multicriteria Decision Support.

1. Introduction

Cloud computing is a new paradigm that provides computing resources as services that can be accessed through the network from anywhere in the world. These resources may include computational processing power and storage capabilities, allowing the elimination of large investments in physical infrastructure characteristic of traditional data center. The NIST (National Institute of Standards and Technology), Mell and Grance (2011), classifies cloud services in three models: SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service). In this work, we will consider the selection of services based on the IaaS model.

The increasing adoption of cloud computing by different types of organizations has given rise to a variety of service types and providers. Each service provider has a diverse range of configurations available for the customer choice, resulting in a large number of services available. As these services have heterogeneous technical and managerial specifications, it is risky to say that a given solution is better or worse than another. Since the breadth of possibilities and different requirements vary for each particular need, such analysis should be based on the different properties of the services available according with the different criteria of a specific need, Armbrust et al. (2009).

*Supported by REUNI-UFRN research scholarship.

The selection of alternatives in the presence of multiple properties or attributes is referred to as a multiple attribute decision making (MADM) problem, Roy (1990). There are several multiple attribute methods that are commonly described in the specialized literature, such as Single Additive Weighting (SAW), Multiple Attribute Value Function (MAVF), Analytic Hierarchy Process (AHP), "Elimination et Choix Traduisant la Realité" (Electre), Preference Ranking Organization Method for Enrichment Evaluations (Promethee), among others. See, for example, Belton and Stewart (2002), Chen and Hwang (1992), for a review of these methods. In general, solving an MADM problem involves sorting or ranking the alternatives, Kahraman (2008)

Therefore, it seems natural to evaluate a set of alternatives related to cloud computing by using multi-criteria decision-making methods. However, generic decision-making methods have to be customized for the domain of IT infrastructure solutions. Customization is non-trivial, given that multiple attribute decision-making methods do not advise on how to find eligible criteria and factors that have an influence on the evaluation and decision making, Menzel et al. (2011). To address this problem, this paper formalizes an unified method for allowing the specification of user requirements for MADM problems in the context of cloud computing. This approach complements previous works published in the fields of Software Engineering (Menzel et al. (2011), Reiff-Marganiec et al. (2009), Sathya et al. (2011)) and Operations Research (Rehman et al. (2012), Garg et al. (2013)), offering an integrated solution for the problem. Our work also has the intent of evaluating the feasibility of the combining multi-criteria methods with the AnaMoC tool, Silva et al. (2011), for selecting cloud computing services. This evaluation has been achieved through a series of experiments involving real cloud computing services from a well-known provider (Amazon).

The remainder of this paper is organized as follows: Section 2 presents some background information about the model defined for specifying user's requirements, the approach adopted for normalizing the different criteria and one of the methods employed. Section 3 describes the methodology adopted in our experiments. The experiments are then described on Section 4 followed by a brief discussion of this exercise. Finally, Section 5 presents some final remarks and some indication of future work.

2. Background

In this section, we present a brief description of the techniques and tools that are being used in our approach. We start by describing a model defined for capturing a selection problem, followed by the strategy used for normalizing the information obtained. Finally, we present a brief description of the Electre I method.

2.1. Describing User Requirements

In the sequel we describe a model that has been created for allowing the specification of user selection criteria in a format that can be interpreted by a computer program. Once the selection criteria are specified based on this model, a normalization procedure can be computerized and used in different multi-criteria methods.

This model, presented in Figure 1 using an UML 2.0 class diagram, contains a root node called **Policy** that represents the objective to be reached by a selection process. A Policy contains a *name* and *version* allowing the identification of one policy from another, and captures the selection requirements in terms of a set of criteria.

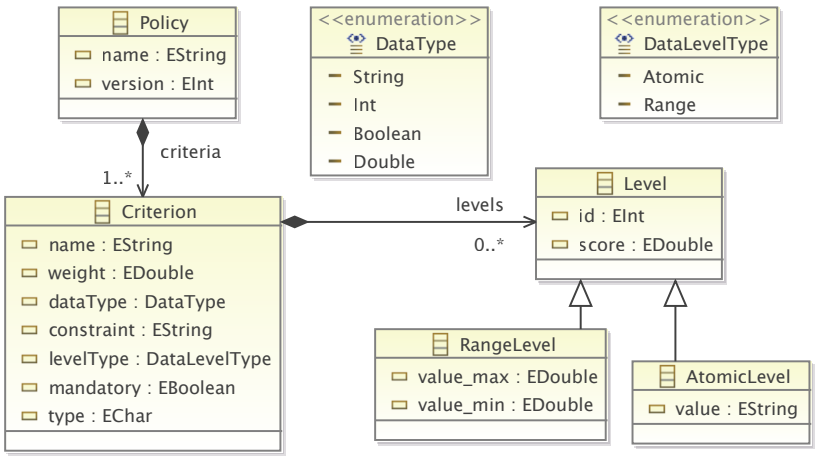

Figure 1. Metamodel used for describing selection policies.

This set of criteria represents the different dimensions of the problem, captured by the **Criterion** element. The attributes of a particular criterion are: *name*: identifier of each criterion; *dataType*: the type of data used in this criterion. For now, we consider the following types: Int, Double, String and Boolean; *weight*: the degree of importance that the user assign to each criterion; *constraint*: a maximum or minimum value allowed to each criterion, this is a special feature of the model to eliminate alternatives that is not in accordance with the user requirements. Its value is manually defined and, if it is omitted, means that any value is acceptable for that criterion; *mandatory*: to indicate whether a given constraint is mandatory or not. In case of a mandatory constraint, any alternative whose property value violates the specified constraint is not considered in the selection. *type*: to indicate if the criterion should be treated as cost (C) or benefit (B).

A criterion can also be organized in levels, where a user can specify different levels of acceptable values (with their associated scores) for the properties of the services being considered during selection. The *levelType* attribute indicates how these levels are interpreted during the selection, identifying how the values will be normalized before fed into a selection method. The designed model defines two ways to represent the levels (as defined by the **DataLevelType** element):

1. **Atomic**: means that the condition for satisfaction of a criterion is associated with a single value, either numeric, Boolean or String types. This attribute is used in four situations: (1) when the value of the *constraint* attribute is omitted; (2) when a value is assigned to the *constraint* attribute indicating which value is expected by the candidate services; (3) when the attribute has a constraints' set of possible values, which is enough for at least one of them be satisfied by the services or; (4) when multiple values are acceptable to a criterion, but organized by **AtomicLevel** element in levels where each level contains a single value;
2. **Range**: means that the conditions for satisfaction of a criterion are represented by numeric values (int or double) and acceptable levels of variability are associated with ranges identified with minimum and maximum values. Each range must have an associated score value, in which case, score values can be explicitly defined or defined based on some technique. The minimum and maximum levels are organized by **RangeLevel** element, where each level contains a range of desirable values;

One possible problem when evaluating services from different providers is the use of different representations by the providers for specifying the same property. For now we assume that the properties from different service instances have the same representation. Thus, we do not need to translate/map to a common representation.

2.2. Normalizing the Obtained Data

The values obtained from the available services for each criterion can be of different types. Moreover, a user may have different requirements that can be represented in the selection policy in different ways. In this way, it is necessary to normalize the values obtained about the available services before applying a multi-criteria decision method. In this section, we describe the approach adopted for automatically normalizing the criteria values based on the services available and on the selection policy defined by the user.

In our work, the normalization is achieved by means of scoring functions, Chakraborty and Yeh (2007), which are responsible for transforming the values obtained from the services available into a score based on the information associated with the selection policy. These scoring functions are selected according with the data type associated with the value, and the format used for representing required values in the policy.

The scoring functions are responsible for calculating the attribute **score** of the element **Level** when it is not manually informed. When this is the case (manually informed scores), the user is responsible for establishing the scores, which should be in a range between zero and one, since all the scores should be normalized. Thus, when the score attribute is empty in a selection policy, our approach is able to find out which scoring function to use based on the criterion data type.

The procedure used to normalize the data are presented in the following:

- **Numerical Type** (**Equation 1**): This equation is employed when the *dataType* attribute is a numerical type (e.g., Int or Double). In this equation v_{max} is the maximum value of all competitive services for the criterion, v is the value for the service under evaluation, and v_{min} is the minimum value of all competitive services.

$$S = \begin{cases} 1 - \left(\dfrac{v_{max} - v}{v_{max} - v_{min}} \right) & \text{if benefit criterion} \\ \left(\dfrac{v_{max} - v}{v_{max} - v_{min}} \right) & \text{if cost criterion} \end{cases} \quad (1)$$

- **Boolean Type** (**Equation 2**): is used for criteria which have a value that is evaluated to true or false.

$$S = \begin{cases} 1 & \text{if criteria is met} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

- **String Type** (**Equation 2**): when the *dataType* attribute is String, Equation 2 is employed using comparison between two texts. This equation is used when the user specifies a constraint, which is compared against the values of this criterion on the alternatives following Equation 2.

2.3. The Electre I Method

The Electre I method is a multiple-attribute model that uses outranking relations with the purpose of finding a set of alternatives dominating over other alternatives, while they cannot be dominated, Wanga and Triantaphyllou (2008). A special feature of this method is the non-compensatory effect, preventing alternatives to be ranked as the best when reaching an excellent score for one or more criteria and, simultaneously, a very low score for another criterion. This particularity avoids undesirable distortions in the final result, ensuring that the alternatives with the best position in the ranking outrank the others. Outranking relations are constructed through two main measures, known as the concordance and discordance indexes, Roy (1990).

According to the classical Electre definition, an outranking relation represents a binary relation over a set of alternatives. Let A_a and A_b be two alternatives for a decision problem, A_a outranks A_b (A_aSA_b) if A_a is, at least as good as A_b, based on arguments derived from the set of evaluation criteria. The main objective is to find the alternatives that dominate over other alternatives without being dominated.

To define an outranking relation it is necessary to calculate two main measures called concordance index [*c(a,b)*] and discordance index [*d(a,b)*]. The concordance index, Roy (1990), can be calculated as follows

$$c(a,b) = \sum_{j=1}^{n} g_j(a,b) w_j, \forall a, b = 1, 2, ..., m, a \neq b \tag{3}$$

where $g_j(a,b)$ is a function that compares the performance rating of alternatives *a* and *b* in relation to the j^{th} evaluation criterion, whose definition is given by

$$g_j(a,b) = \begin{cases} 1.0, & \text{if} \quad r_{aj} > r_{bj}, \\ 0.5, & \text{if} \quad r_{aj} = r_{bj}, \\ 0.0, & \text{otherwise} \end{cases} \tag{4}$$

The concordance index measures the strength of the hypothesis of a given alternative A_a, that is at least as good as alternative A_b. On the other hand, the discordance index measures the strength of the evidence against this first hypothesis, Wanga and Triantaphyllou (2008). The discordance index, Roy (1990), can be calculated as follows

$$d(a,b) = \begin{cases} 0, & \text{if} \quad r_{aj} \geq r_{bj}, j = 1, ..., m \\ \frac{max_{j*}(r_{bj} - r_{aj})}{max_{j=1...m}(|r_{bj} - r_{aj}|)}, & \text{otherwise} \end{cases} \tag{5}$$

Where set $j^* = \{j \mid r_{bj} \geq r_{aj}\}$ contains the index of all criteria against the assertion "A_a is at least as good as A_b". Once these two indexes were define, a binary outranking relation S can be defined as

$$A_a S A_b \quad iff \quad c(a,b) \geq \overline{C} \quad and \quad d(a,b) \leq \overline{D} \tag{6}$$

where $\overline{C}$ and $\overline{D}$ are thresholds defined by the decision makers. The thresholds can be adjusted in order to provide more helpful information to decision-makers, according to

their preferences within a sensitivity analysis study.

After calculating the concordance and discordance matrices, a directed graph, called outranking graph, can be drawn with the purpose of representing the outranking relations between all alternatives. In this graph, the nodes represent the alternatives and each directed arc indicates that an alternative dominates over another. The outranking graph can be mathematically defined as a directed acyclic graph $G = (A, S)$, where the vertices correspond to the alternatives and the arcs correspond to the binary outranking relations defined in $S[s_{ab}]_m \times_m$, where s_{ab} = 1 if $A_a \mathrm{S} A_b$, and s_{ab} = 0, otherwise. The outranking graph provides a partial ranking of the alternatives. Additionally, the set of nodes with in-degree equals to zero provides the set of dominant alternatives.

Electre I provides a partial ranking and a set of promising alternatives. When a full ranking of the alternatives is required, different extension of the Electre may be used. Chatterjee et al. (2010) presented two measures, called pure concordance index (PCI) and pure discordance index (PDI), which can be utilized to rank a set of the alternatives in Electre models. These measures are defined as follow

$$PCI_i = \sum_{j=1}^{m} c(i,j) - \sum_{j=1}^{m} c(j,i) \quad i = 1, ..., n, \quad and \quad j \neq i \tag{7}$$

$$PDI_i = \sum_{j=1}^{m} d(i,j) - \sum_{j=1}^{m} d(j,i) \quad i = 1, ..., n, \quad and \quad j \neq i \tag{8}$$

From these measures, the set of alternatives must be sorted in descending order by *PCI* and in ascending order by *PDI*. The final position of each alternative is given by the average position of each partial ranking.

3. Methodology

In this section, we describe the application methodology of our approach in terms of a service selection process that has been defined by us. We start by presenting a modular view of the whole selection process, which is then followed by its detailed description.

3.1. A Modular View of the Selection Process

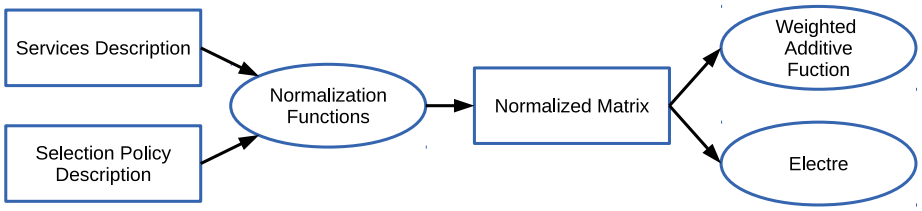

Figure 2. General view of the adopted methodology.

Figure 2 presents a general view of our approach. In this Figure, *Services Description* corresponds to a description of all available services and their respective properties in a format suitable to be processed by our tool. The *Selection Policy Description* represents the user requirements expressed by means of a selection policy based on the model presented on Section 2.1. The *Normalization Functions* corresponds to the application of the

scoring functions of Section 2.2 taking into the account the descriptions previously mentioned. The result of applying the normalization functions is a *Normalized Matrix* considering the different criteria and their respective weights. The normalized matrix is then fed into the methods that we are currently evaluating, namely, a weighted additive function and the Electre method. Finally, the results of both methods are compared and evaluated.

3.2. Steps of Service Selection Process

In the sequel, we detail the steps of the process defined for selecting services, which is presented in Figure 3.

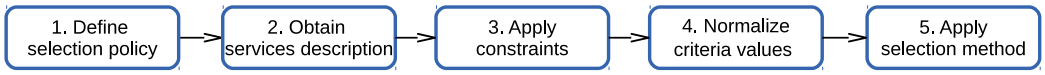

Figure 3. The service selection process.

Step 1: Define the selection policy. The selection policy corresponds to the hierarchical structure of the problem. The definition of the criteria is one of the most difficult tasks in any decision making process. Although there is no universal method described in the literature to generate and structure a set of criteria, Keeney and Raiffa (1976) points out important evaluation aspects when carrying out this task as follows: (i) criteria should be meaningful indicators of performance; (ii) the definition of the attribute should be concise and clear; (iii) the set of criteria must address all the critical aspects of a problem; and (iv) the set of criteria should be defined in a such a way that the same dimension is not measured by several different and independent criteria in the criteria structure.

Step 2: Obtain the description of the available services. This step consists in obtaining information about the available services and their respective properties. These descriptions are stored in a repository and are updated based on the service provider Web site.

Step 3: This step of the process consists in eliminating any service that does not satisfy a constraint for a mandatory criterion. Thus, our module compares the value of that criterion from the service with the value set in the policy, and if the service does not satisfy the policy, the service is discarded.

Step 4: Normalize the rating of the alternatives i for each evaluation criterion j in order to generate the matrix $D = [r_{ij}]_{m} \times_{n}$ by using the scoring functions described in section 2.2; For those services that satisfied the mandatory criteria, the module will calculate a score for each criteria. To do that, the module considers the value type of that criterion according to the attribute levelType and applies the scoring functions depending on the specified selection policy:

1. If the value of attribute levelType is Range, the value of score manually entered by the user is used;
2. If the value of attribute levelType is Atomic, then according to the data type of that criterion one of the scoring functions will be used. If the type is int or double, equation 1 is used, otherwise (if the type is boolean or String), equation 2 is used.

Step 5: Apply multi-criteria selection method. After each criterion has its value normalized, a multicriteria selection method is applied. Currently we are emplying a weighted additive function and the Electre method (which has been described on Section 2.3). The weighted additive function simply sums the different criteria values multiplied by their res-

pective weights in order to establish a rank for each alternative. The Electre method has been implemented in the following way:

1. Construct the weighted decision matrix $D = [r_{ij}] \times [w_j]$;
2. Calculate the concordance index as Equation 3;
3. Calculate the discordance index as Equation 5;
4. Calculate the pure concordance and pure discordance indexes as Equations 7 and 8, respectively;
5. Calculate the final ranking of the alternatives by using the concordance and discordance index previously computed.

This process has been implemented in a tool using the Java programming language. The Selection Criteria Description follows the model presented in Section 2, while the Universal Service Description Language (USDL), Cardoso et al. (2010), has been used for representing Services Description. The scoring functions used for normalizing the values have been adapted from the AnaMoC, Silva et al. (2011), a tool for selecting software components in the context of self-adaptive software systems. The two multi-criteria methods considered were implemented as components that have been plugged into our tool.

4. Case Study

This section presents a case study that has been used for experimenting with our service selection process and its implementation. We start by describing the application of our service selection process, which is followed by a brief discussion of this exercise.

In this case study, we have considered a set of services offered by a real cloud provider, Amazon[1], which is today one of the most complete providers of computational services of the market. These services are categorised as *On-Demand Instances*, which let you pay for computing capacity by the hour with no long-term commitments. In this category, a user can choose from a number of different instance types to meet its computing needs (e.g., Standard Instances, Micro Instances, High-Memory Instances, High-CPU Instances). Each instance provides a predictable amount of dedicated compute capacity and a set of properties that can be configured according with the specific need of the user.

Based on the services provided by Amazon, we have applied our service selection process, which is described below.

Step 1: Following the steps defined in our service selection process, it is necessary to define the selection policy with the different criteria and associated weights capturing the user requirements. Based on the properties of the services available, and in interviews with experts from the industry, we have identified eight criteria as follows: CPU (C1), RAM (C2), Plataform (C3), Storage (C4), I/O Performance (C5), EBS-Optimezed Available (C6), Cost (C7) and Operating System - OS (C8). The selection policy established for our experiments is presented on Table 1.

As presented on Section 2, the selection policy must be specified according to our model so it can be used by our tool. Criteria C1, C3 and C5, was modeled as DataType String and organized by AtomicLevel, where the acceptable values are associated with a score that is manually defined. In this example, we assign growing priority (score) to the extent that the amount of the criterion value increases. Criteria C2 and C4, use Double and

[1] http://aws.amazon.com/

Table 1. Selection policy with the rules applied to each criteria.

Criterion	Name	Weight	type	DataType	Constraint	Dimension	Mandatory	Level			
								AtomicLevel		RangeLevel	
								Value	Score	Value	Score
C1	CPU	0.25	B	String	-	Atomic	false	Up to 2 EC2 1 EC2 2 EC2 4 EC2 5 EC2 6.5 EC2 8 EC2 13 EC2 20 EC2 26 EC2	0.1 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9	-	-
C2	RAM	0.2	B	Double	-	Range	false	-	-	0-1.15 1.16-5.4 5.5-11.25 11.26-23.55 23.56-51.3 51.4-∞	0.2 0.4 0.5 0.6 0.7 0.8
C3	Platform	0.1	B	String	-	Atomic	false	32-bit 64-bit	0.3 0.5	-	-
C4	Storage	0.15	B	Int	-	Range	false	-	-	0-380 381-635 636-∞	0.2 0.3 0.4
C5	I/O Perf	0.05	B	String	-	Atomic	false	Low Moderate High	0.1 0.2 0.3	-	-
C6	EBS	0.05	B	Boolean	true	Atomic	false	-	-	-	-
C7	Cost	0.2	C	Double	1.0	Atomic	false	-	-	-	-
C8	OS	0.0	B	String	Linux/UNIX	Atomic	true	-	-	-	-

Int as DataType, respectively, and are organized by RangeLevel, where different ranges of values are given a determined score. Criteria C6, C7 and C8 employ constraints, which restrict in different ways the services to be considered and the score value achieved. Criteria C6 and C7 have their mandatory flag set to false, which means that whenever a property does not satisfy a given constraint, it will receive score zero. On the other hand, criterion C8, a mandatory criterion, is used to filter out those service that do not satisfy the constraint, reducing the number of alternatives considered by the selection method further on.

Step 2: The second step is to obtain the description of services and their properties from the cloud service provide. In this case study we have identified a total of 32 services considering the different instance types provided by Amazon, and configuration values for the services properties.

Step 3: In this step, there is an elimination of those services which contain a property that does not satisfy a mandatory criterion. In this experiment, we choose the operating system as mandatory criterion, with two possibilities available (Linux and Windows). We have defined Linux as a constraint for this criteria, and thus, our tool eliminates all alternatives whose operating system property is Windows. Table 2 describes the services that satisfied this constraint. Notice that services 3, 4, 7, 8, 10, 12, 14, 16, 19, 20, 22, 24, 26, 29, 30 and 32 do not appear on the table, as they had the criterion C8 set to Windows;

Step 4: Once the services that do not satisfy a mandatory criterion have been filtered out, the next step is a normalization of the properties of the remaining services. For that, the scoring functions of Section 2.2 are applied to the different criteria. For those criteria that have manually defined scores (C1, C2, C3, C4, and C5), the levelType attribute is used to

Table 2. Computing resources configurations and prices (Services description).

Service	C1	C2	C3	C4	C5	C6	C7
1	1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)	1.7 GB	32-bit	160 GB	Moderate	No	$0.080/hour
2	1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)	1.7 GB	64-bit	160 GB	Moderate	No	$0.080/hour
5	2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit)	3.75 GB	32-bit	410 GB	Moderate	No	$0.160/hour
6	2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit)	3.75 GB	64-bit	410 GB	Moderate	No	$0.160/hour
9	4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)	7.5 GB	64-bit	850 GB	Moderate	500 Mbps	$0.320/hour
11	8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each)	15 GB	64-bit	1690 GB	High	1000 Mbps	$0.640/hour
13	13 EC2 Compute Units (4 virtual cores with 3.25 EC2 Compute Units each)	15 GB	64-bit	EBS only	Moderate	500 Mbps	$0.680/hour
15	26 EC2 Compute Units (8 virtual cores with 3.25 EC2 Compute Units each)	30 GB	64-bit	EBS only	High	1000 Mbps	$1.360/hour
17	Up to 2 EC2 Compute Units (for short periodic bursts)	613 MB	32-bit	EBS only	Low	No	$0.027/hour
18	Up to 2 EC2 Compute Units (for short periodic bursts)	613 MB	64-bit	EBS only	Low	No	$0.027/hour
21	6.5 EC2 Compute Units (2 virtual cores with 3.25 EC2 Compute Units each)	17.1 GB	64-bit	420 GB	Moderate	No	$0.540/hour
23	13 EC2 Compute Units (4 virtual cores with 3.25 EC2 Compute Units each)	34.2 GB	64-bit	850 GB	High	500 Mbps	$1.080/hour
25	26 EC2 Compute Units (8 virtual cores with 3.25 EC2 Compute Units each)	68.4 GB	64-bit	1690 GB	High	1000 Mbps	$2.160/hour
27	5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each)	1.7 GB	32-bit	350 GB	Moderate	No	$0.200/hour
28	5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each)	1.7 GB	64-bit	350 GB	Moderate	No	$0.200/hour
31	20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each)	7 GB	64-bit	1690 GB	High	1000 Mbps	$0.800/hour

decide the score of each property. On the other hand, criteria C6, C7 and C8 have their score dynamically calculated. Criterion C7, which determines the cost, has a constraint value of 1, but since it is not a mandatory value, those services that violate the constraint are still considered in the selection process, but with a score of zero. Table 3 shows the normalized decision matrix after the application of the scoring functions.

Table 3. Normalized decision matrix.

Service	C1	C2	C3	C4	C5	C6	C7
1	0.1	0.4	0.3	0.2	0.0	0.0	0.975
2	0.1	0.4	0.5	0.2	0.0	0.0	0.975
3	0.0	0.4	0.3	0.3	0.0	0.0	0.975
4	0.0	0.4	0.5	0.3	0.0	0.0	0.975
5	0.3	0.5	0.5	0.4	0.0	0.1	0.975
6	0.6	0.6	0.5	0.4	0.3	1.0	0.975
7	0.7	0.6	0.5	0.2	0.0	1.0	0.975
8	0.9	0.7	0.5	0.2	0.3	1.0	0.0
9	0.1	0.2	0.3	0.2	0.1	0.0	0.975
10	0.1	0.2	0.5	0.2	0.1	0.0	0.975
11	0.5	0.6	0.5	0.3	0.0	0.0	0.975
12	0.7	0.7	0.5	0.4	0.3	1.0	0.0
13	0.9	0.8	0.5	0.4	0.3	1.0	0.0
14	0.4	0.4	0.3	0.2	0.0	0.0	0.975
15	0.4	0.4	0.5	0.2	0.0	0.0	0.975
16	0.8	0.5	0.5	0.4	0.3	1.0	0.975

Step 5: The normalised matrix is then used as input for the selection methods being consi-

dered, namely, Electre and weighted additive function. According with the description of the Electre method, the concordance and discordance indices are computed, and then used to calculate the pure concordance and pure discordance indices. Those are presented on Table 4, together with the final ranking of the alternatives for both methods.

Table 4. Final Ranking of the alternatives.

	Electre						Weighted Additive Function	
Service	PCI	PDI	Rank PC	Rank PD	Average Rank	Final Rank	WAF Rank	Final Rank
1	-5.70	4.475	15.0	14.0	14.5	11	0.36	13
2	-4.09	3.575	12.0	11.0	11.5	9	0.38	11
3	-5.55	4.575	14.0	16.0	15.0	12	0.35	14
4	-3.94	3.775	11.0	12.0	11.5	9	0.37	12
5	2.10	0.775	8.0	5.0	6.5	6	0.485	8
6	6.35	-12.225	3.0	2.0	2.5	2	0.64	2
7	3.35	-10.825	6.0	3.0	4.5	4	0.62	3
8	3.55	2.175	5.0	8.0	6.5	6	0.51	6
9	-6.75	4.475	16.0	15.0	15.5	12	0.325	16
10	-5.15	3.775	13.0	13.0	13.0	10	0.345	15
11	2.90	-0.025	7.0	4.0	5.5	5	0.535	5
12	4.90	2.375	4.0	9.0	6.5	6	0.49	7
13	7.0	1.475	1.0	6.0	3.5	3	0.56	4
14	-3.7	2.375	10.0	10.0	10.0	8	0.435	10
15	-2.10	1.575	9.0	7.0	8.0	7	0.455	9
16	6.85	-12.325	2.0	1.0	1.5	1	0.67	1

Before making comments about the experiments, it is very important to point out that we assumed that there is no optimal method for a MADM problem. Table 4 shows some dissimilarities between the rankings provided by weighted additive function (WAF) and by Electre method. Differences in the rankings were expected, since the introduction of the concordance and discordance indexes significantly affects the way the final ranking of the alternatives is computed, and the Electre is a non-compensatory method, while the WAF is a compensatory method. However, both methods elected the alternatives 9 and 10 as the two best, and the alternatives 1, 3, 9 and 10 as the four worst.

Finally, the proposed methodology used to generate the results showed that MCDM can be efficiently used for cloud service selection, but they also showed that a hard work must be done in order to specify tools and models to capture the user requirements, collect the data and transform the various criteria values into a comparable scale. The researches on traditional MCDM methods have been neglecting these issues.

5. Conclusions

This paper has presented an approach for the selection of cloud computing services based on MADM methods. We have defined an integrated solution that supports a complete service selection process, from the description of a user requirements and of the services available with their respective properties, to the normalization of the values of the different criteria, and the application of different MADM methods. Our solution complements the works in the field of Software Engineering and Operational Research in a tool that has been developed by us and used to conduct some experiments.

As future work, we plan to refine the model for description of user requirements, and to explore it for selection problems in other contexts for evaluating its flexibility. We also intent to incorporate other normalization procedures, and other MADM methods into our tool, providing an environment that can be used for conducting operation research experiments based on principles of software engineering.

References

Armbrust, M. et al. (2009). Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Belton, V. and Stewart, T. J. (2002). *Multiple criteria decision analysis: an integrated approach*. Klewer Academic Publishers, Boston.

Cardoso, J. et al. (2010). Towards a unified service description language for the internet of services: Requirements and first developments. In *Proceedings of the IEEE International Conference on Services Computing (SCC)*, pages 602–609.

Chakraborty, S. and Yeh, C.-H. (2007). A simulation based comparative study of normalization procedures in multiattribute decision making. In *Proceedings of the 6th WSEAS Int. Conf. on Artificial Intelligence*, pages 16–19.

Chatterjee, P., Athawaleb, V. M., and Chakraborty, S. (2010). Selection of industrial robots using compromise ranking and outranking methods. *Robotics and Computer-Integrated Manufacturing*, 26:483–489.

Chen, S.-J. J. and Hwang, C. L. (1992). *Fuzzy Multiple Attribute Decision Making: Methods and Applications*, volume 375 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag.

Garg, S. K., Versteeg, S., and Buyya, R. (2013). A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4):1012 – 1023. Special Section: Utility and Cloud Computing.

Kahraman, C. (2008). Multi-criteria decision making methods and fuzzy sets. In *Fuzzy Multi-Criteria Decision Making: Theory and Application with Recent Developments*, volume 16 of *Springer Optimization and Its Applications*, pages 1–18. Springer.

Keeney, R. L. and Raiffa, H. (1976). *Decisions with multiple objectives: Preferences and value tradeoffs*. J. Wiley, New York.

Mell, P. and Grance, T. (2011). *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology.

Menzel, M., Schönherr, M., and Tai, S. (2011). (mc2)2: criteria, requirements and a software prototype for cloud infrastructure decisions. *Software: Practice and Experience*, pages 1–15.

Rehman, Z., Hussain, O. K., and Hussain, F. K. (2012). Iaas cloud selection using mcdm methods. In *ICEBE*, pages 246–251.

Reiff-Marganiec, S., Yu, H. Q., and Tilly, M. (2009). Service selection based on non-functional properties. In *Service-Oriented Computing - ICSOC 2007 Workshops*, pages 128 – 138. Springer-Verlag.

Roy, B. (1990). *Readings in multiple criteria decision aid*, chapter The outranking approach and the foundations of ELECTRE methods. Springer-Verlag.

Sathya, M. et al. (2011). Evaluation of qos based web-service selection techniques for service composition. *International Journal of Software Engineering*, Vol. 1 Issue 5:73–90.

Silva, D. C. D. et al. (2011). Definition of a component selection process based on qos criteria and its application to self-adaptive software systems. *Fifth Brazilian Symposium on Software Components Architectures and Reuse*, pages 90–99.

Wanga, X. and Triantaphyllou, E. (2008). Ranking irregularities when evaluating alternatives by using some electre methods. *Omega*, 36:45–63.