Soltani et al. Journal of Cloud Computing: Advances, Systems and Applications (2018) 7:12
https://doi.org/10.1186/s13677-018-0113-8

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

A hybrid approach to automatic IaaS service selection

Sima Soltani[1*], Patrick Martin[1] and Khalid Elgazzar[2]

Abstract

Cloud computing provides on-demand resources and removes the boundaries of resources' physical locations. By providing virtualized computing resources in an elastic manner over the internet, IaaS providers allow organizations to save upfront infrastructure costs and focus on features that discriminate their businesses. The growing number of providers makes manual selection of the most suitable configuration of IaaS resources, or *IaaS services*, difficult and time consuming while requiring a high level of expertise. In our previous paper we proposed QuARAM recommender, a general platform for automatic IaaS service selection. In this paper, we present in detail the hybrid approach to automatic service selection used in our platform. The selection process begins with automatic extraction of an application's features, requirements and preferences, which are then used to produce a list of potential services for the application's deployment. We use case-based reasoning and MCDM (Multi-criteria Decision Making) to provide a recommendation of suitable services for application deployment, clustering to handle the problem of a large search space and a service consolidation method to improve the resource utilization and decrease the total service price. We carry out a case study with a prototype implementation of our platform to demonstrate that automatic IaaS service selection using a combination of all the proposed approaches is both practical and achievable.

Keywords: Cloud computing, Service selection, Case-based reasoning, Multi-criteria decision making

Introduction

Cloud computing has become a technology that affects many aspects of our everyday life. Organizations started to adopt cloud computing as an approach to augment, or even entirely replace, their existing IT infrastructure. As a result, many organizations are considering moving their applications and data to a cloud environment in order to take advantage of its flexibility and potential cost savings [1]. It is anticipated that the fluidity of, and the competition within, the cloud computing market will grow as the technology matures. This will encourage providers to adopt a wider range of mechanisms, such as discounts and incentives, to attract potential consumers [2].

There are currently more than 60 public IaaS providers [3] who offer a variety of services, that is configurations of VMs, storage, and bandwidth, across the globe. Competing providers attract customers by improving their performance while lowering their prices. Although this large number of services provides customers with the opportunity to focus on their core business and revenue growth, it makes the decision of selecting a suitable combination of services challenging. Such a decision must be made based on meeting both the application's requirements and the customers' goals, desires and constraints. Cloud customers need a system to help them in deciding what services to select in order to optimize their resource allocation.

Consider a simple application that needs to be deployed on a public cloud. Figure 1 illustrates the minimum system requirements (CPU, RAM, storage, OS) of the different components of this application, the QoS requirements (availability, security) and application requirements (maximum concurrent users, region, and bandwidth). The number to the right of each feature represents the customer's respective preferences, ranging from 0 to 10, where 0 means "no preference" and 10 means "most required". Given the requirements

* Correspondence: soltani@cs.queensu.ca
[1]School of Computing, Queen's University, Kingston, Canada
Full list of author information is available at the end of the article

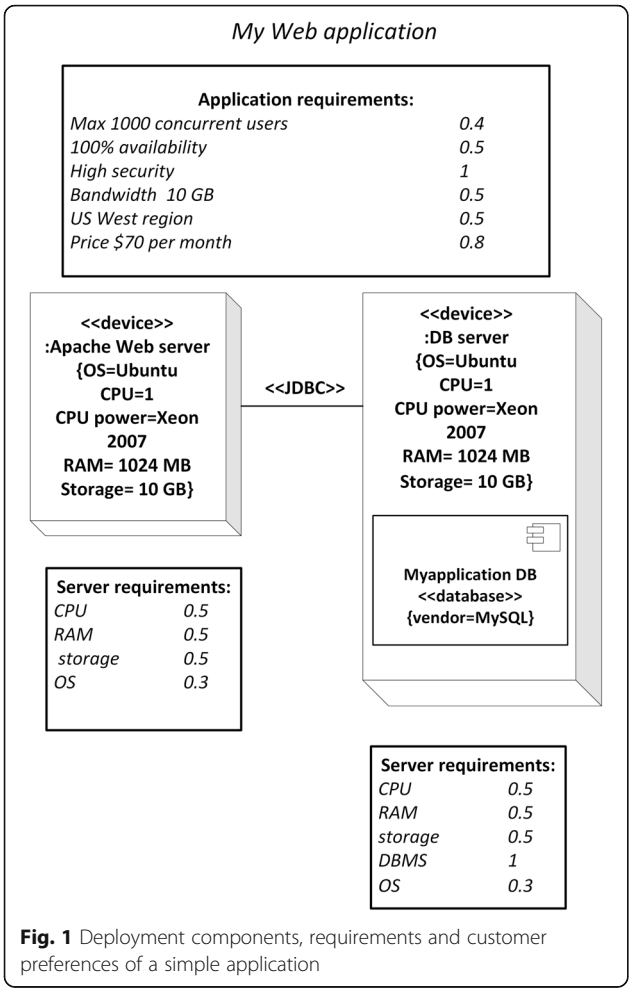

Fig. 1 Deployment components, requirements and customer preferences of a simple application

described for the application, it can be deployed on two VM instances, one for the Apache Webserver and the other one for the MySQL database server.

There are more than 20 different IaaS providers who can satisfy the requirements and provision service (VMs) to these two server instances. In 2018, for placing each component on a separate VM the price varies from $10 to $98 for each instance (e.g., eApps: Basic instance, Joyent Cloud: Hardware VM standard 1.75 GB RAM, Rackspace: 1–2 SSD instance). Another option is to deploy the two servers on a single instance (e.g., eApps: Advance instance or Rackspace: General 1–4 SSD instance) [4]. The decision on the appropriate service is not simply based on the service price, but also must take into consideration the service performance and the experience of previous customers.

Previous studies show that due to the large size of the search space of available services, and the wide range of heterogeneous selection criteria used by customers, there is a need for a robust approach that can reduce the complexity of the search by either reducing the search space or the number of comparisons required to select the appropriate service [5]. The approach should support both qualitative and quantitative criteria.

To develop an automatic cloud service selection system, three challenges need to be addressed:

1. **Automatic identification and extraction of application requirements and customer preferences:** The application requirements and customer preferences should be automatically extracted from the application description document. This includes defining new descriptive features of application requirements that can be used in decision making. These features must therefore be incorporated into the specification of cloud applications.
2. **Automatic evaluation, selection and integration of services:** Lacking the proper level of experience in the field of cloud services makes it difficult for customers to select the best deployment plan for their applications on the cloud. Moreover, providing them with too many options makes it even more confusing. The decision must balance several factors including the application's requirements, the customer's goals, desires and constraints, price and performance. This wide range of heterogeneous selection criteria and the large search space of available services mandate the need for an automatic and robust approach that simplifies cloud service selection. A comprehensive approach must consider all determining attributes (both quantitative and qualitative) to present the best options to consumers. The approach should be able to handle the large number of alternatives as well as missing values for some of the attributes. Medium and large size applications are likely to need multiple services for deployments. In fact, for many applications, it is more cost effective if they are deployed on multiple small services (i.e., split the application into separate deployment entities and deploy each one separately) instead of one large service. This is especially in cases a deployed large service is under-utilized by an application (i.e., the customer pays for resources that are not used by the application). Service integration is therefore an additional challenge that service selection needs to carefully handle. In addition, the deployment region is an important attribute in the application deployment to facilitate communications between its different components. Another important characteristic of the approach that can affect the performance of the service selection system is to be able to remember previous deployments in order to reduce the number of searches and the response time.

3. **Adaptation to dynamically changing environment:** The cloud is highly dynamic and the status of its services is continuously changing (e.g., change in price, or QoS). A robust system must have the ability to adapt to these changes to persistently meet the customer's satisfaction. Otherwise, the service reputation will be compromised and customers will provide negative feedback and low ratings. The service selection system must incorporate the customer's feedback as well as feedback from monitoring systems to support better recommendations.

In this paper, we describe in detail the models and techniques used in QuARAM Service Recommender to address the aforementioned challenges in recommending suitable services for cloud application deployments and extend our previous work on a platform for service selection [6]. The platform is a part of the QuARAM (QoS-aware cloud application management) framework [7] (Fig. 2). This autonomic framework facilitates selecting an appropriate cloud provider, provisioning resources on that provider, deploying the application on those resources, monitoring the execution of application, and dealing with performance challenges and errors that may arise. The QuARAM Service Recommender provides the *Deployment Engine* and *Recommender* components of the QuARAM framework. The *Deployment Engine* is responsible for parsing and extracting information from the specification of applications and interaction with the customer in selection process. The Recommender component is responsible for the rest of service recommendation process.

The QuARAM Service Recommender platform is a comprehensive extension to our previous work [2, 6]. In that previous work, we described a recommendation service for an application using the case-based

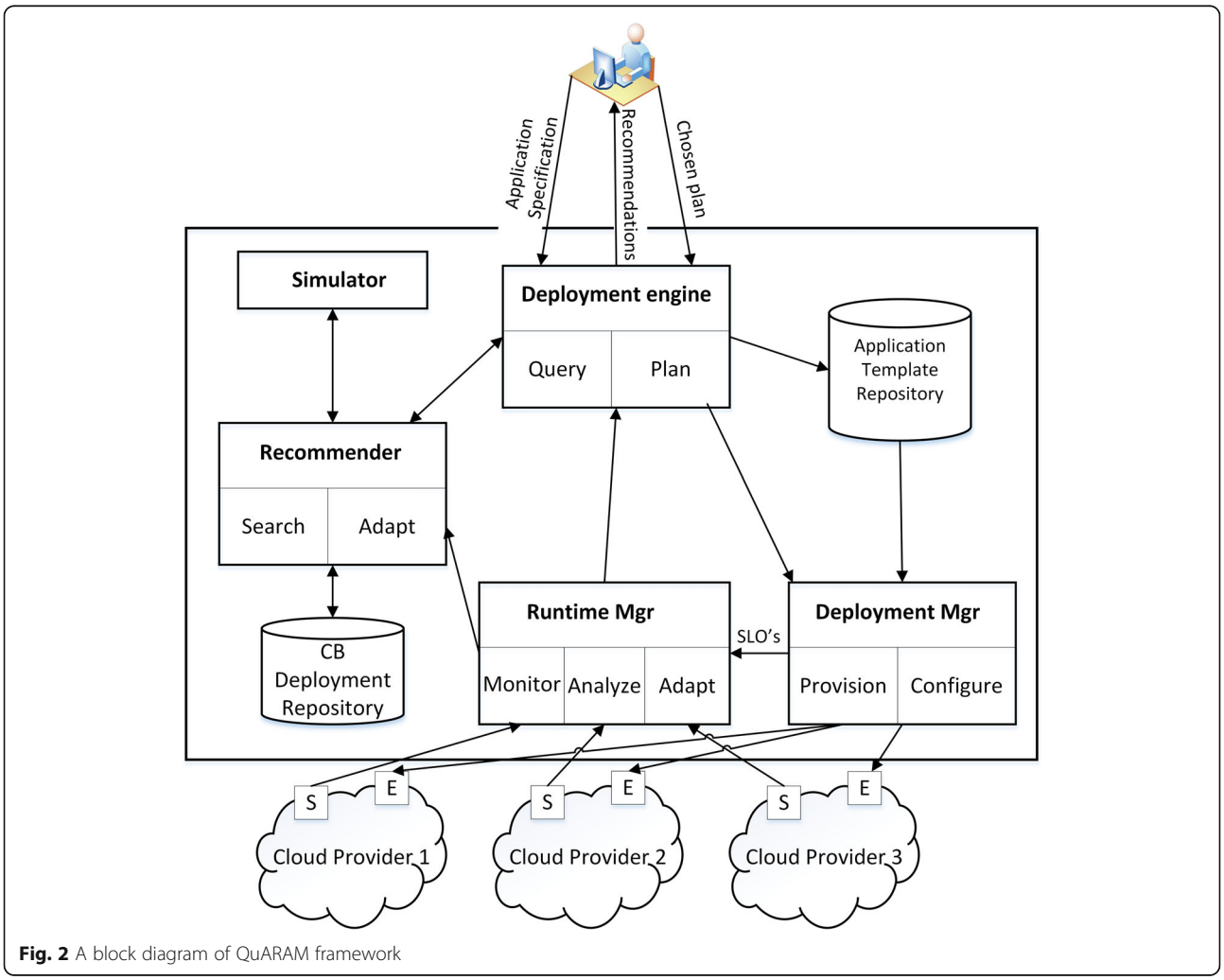

Fig. 2 A block diagram of QuARAM framework

reasoning approach. In this paper, we present three extensions to our initial system and demonstrate our comprehensive platform that encompass all the components for service selection.

First, our case-based reasoning approach to service selection requires cases in the case base that are similar to the target case. We consider the situation when there are no similar experiences in the case base. We therefore need to find providers that can supply the required functionality for the entire application (or deployment entities) without the case-based recommender system. We propose to use Multi-Criteria Decision-Making (MCDM), and specifically the TOPSIS method, to rank the available offerings.

Second, we consider the scenario where the VMs selected by our service are underutilized. We need to increase the resource utilization, while maintaining or decreasing the total application deployment price. We propose a method for service consolidation to reach this objective.

Third, as mentioned in third challenge above, the system needs to incorporate feedback from the customer and monitoring system to be able to provide better recommendations. We describe how reinforcement learning is used in our platform to improve the accuracy of the recommendation system using customer and monitoring feedback.

The remainder of the paper is organized as follows. Related work section describes related work and their relationship to our work. Hybrid service selection section describes in detail the use of MCDM for service selection. We present service consolidation method for better utilization and lowering the price in Service consolidation section. An overview of the QuARAM Service Recommender architecture along with a detailed description of the platform components and the recommendation procedure is presented in QuARAM service recommender architecture section. Validation of the QuARAM service recommender section provides a validation of our approach with a case study involving a proof-of-concept implementation and a step-by-step process of service selection. Finally, Summary section concludes the paper and highlights future research directions.

Related work

Several studies have addressed the selection of IaaS cloud services. CSRS [8] is a conceptual framework that compares all available cloud services based on the performance of virtual machines, QoS and users' feedback. This framework selects services that satisfy the technical requirements of the application first and then eliminates those that do not accommodate the cost constraints of the customer. The selected set is then ranked and presented to the customer. Zang et al. [9] propose a two-step approach that uses a maximum gain and minimum cost to optimize the service selection. In this proposed algorithm, first available services are selected and then based on the defined gain and cost of the services, a service is recommended to the user. Rehman et al. [10] introduce a mathematical formulation and method based on a set of abstract criteria for selecting a cloud service provider. The authors pointed out that this method is only effective for service selection amongst offerings that have similar specifications but different performance. All these approaches entail high complexity as they compare all available cloud services against all criteria. They also do not scale well, given that the number of services is increasing dramatically.

Chen et al. [11] utilize Constraint Programming (CP) to solve the problem of service selection in the cloud. Although the authors aim at reducing the number of comparisons, the complexity of the algorithm is still high when the numbers of constraints and services are large. Their approach also does not accommodate preferences of the user over different requirements.

Another group of studies define service selection as a multi-criteria decision making (MCDM) problem [12]. Godse et al. [13] employ the Analytical Hierarchy Process (AHP) method to provide SaaS cloud service selection. The drawback of this method is its limitation with respect to the number of alternatives with multiple criteria [5]. Chung et al. [14] used ANP (Analytic Network Process) for service selection. They suggest a set of high level criteria for cloud service selection and use a survey of CIO, CEO, and ICT experts to determine the importance of each criteria. The CloudRecommender [5, 15] cloud service selection system defines cloud service selection as a multi-criteria optimization problem. It defines ontologies for service and QoS to facilitate the discovery of services based on their functionality and QoS parameters. The system solves the problem with genetic algorithms and AHP as the fitness function to handle mixed quantitative and qualitative criteria. Qian et al. [16] propose a heuristic approach to search the solution space. Their approach organizes the selection problem as a graph that encompasses the components of the target application and the application's potential clients. It then finds a service for each node in the graph based on the service cost, distance to clients, distance to other components of the application, and the reliability of the provider. Lee et al. [17] proposed a hybrid MCDM model focused on IaaS service selection for firms' users that is based on balanced scorecard (BSC), fuzzy Delphi method (FDM) and fuzzy AHP. BSC is used to prepare

a list of decision making factors. FDM is used to select the list of important decision-making factors based on the decision makers' opinion (using a questionnaire) and FAHP is used to rank and select the best cloud service. This work's focus is on the migration of the whole company ICT to cloud based on a set of general cloud service features. Baranwal et al. [18] propose to use Improved Ranked Voting Method (IRVM). In this method, the services are compared and ranked first based on each metric individually and then the services are scored based on the preference of the customer and the ranking in the previous step. In all of the above methods, the decision is made by comparing and ranking all of the available services. With the rapid growth of the cloud industry, there would be a large search space problem for all of these methods.

Sunderes et al. [19] tackle the problem of a large search space through indexing the cloud services (using iDistance techniques) based on their common set of properties. This approach first indexes cloud providers based on their properties in a B^+-tree, and then uses the K-NN algorithm (K-Nearest Neighbor) to find the service that fits the requirements of the customer. Since a cloud service user may have a set of service requirements that cannot be fulfilled by any single service provider, they propose to aggregate service providers. If the result of the search satisfies all the requirements, then it is recommended to the user. Otherwise, if only some of the requirements are satisfied, a new query is issued to satisfy the remaining requirements. The process continues until all the requirements are satisfied.

We identify two shortcomings in service selection approaches in the literature. They reduce the number of comparisons by either using heuristic algorithms or indexing the search space. However, they do not accommodate the customers' preferences in the decision making. Furthermore, all approaches handle every application deployment as a new case without taking into account the results of similar previous deployments.

Hybrid service selection

In order to address the second challenge in providing automatic service selection (automatic evaluation, selection and integration of services) we primarily use case-based reasoning for evaluation and selection of services that can accommodate the application [2]. We incorporate the application's requirements, and the customer's priorities in our decision making using a set of similarity measures and weights in similarity calculation. In case-based reasoning system, previously deployed applications are used to help in selection of services for new applications. Using a set of similarity functions (local and global similarity functions) the similarity of the target application's requirements and preferences to the applications already in the case base is calculated and the top most similar ones are used to recommend a list of solutions (services) for the target application. The discovered solutions are then adapted to the requirements and preferences of the target application.

To use case-based recommendation for service selection, it is required to have cases in the case base that are similar to the target case. Sometimes there are no similar cases available in the case base. In this situation, we need to find providers that can supply the required functionality for the entire application without the case-based recommender subsystem.

A cloud application consists of a set of deployment entities. For example, the application in Fig. 1 has two deployment entities. For each of the deployment entities, the search for the suitable service can be performed using the case-based recommender. If no similar deployment entities are found in the case base (i.e., the similarity of all cases is less than a specified threshold), then the recommendation system searches for a suitable service among all available services that are offered by the cloud providers.

The number of possible services available from IaaS providers for deployment of entities is large and growing rapidly. We use clustering to decrease this search space and improve the overall response time, while maintaining high precision. We use the k-means clustering algorithm [20] to build a model to cluster all available services based on their advertised features. The model is then used to find the cluster to which a deployment entity belongs. The search engine aims to find the service in that cluster from different providers that best satisfies an entity's requirements. Within the cluster, we use the K-Nearest Neighbor (K-NN) algorithm [21] to find the most similar service to the target entity. Customer preferences for different requirements are represented as weights in the similarity function, which is given as:

$$Sim(service, deployment\ entity) = \frac{\sum_{i=1}^{n} w_i \times Sim\left(f_i^s, f_i^d\right)}{\sum_{i=1}^{n} w_i} \quad (1)$$

where w_i is the weight[1] of the i^{th} requirement of the application entity, f_i^d is the application's required value for the i^{th} requirement and f_i^s is the value for the service's corresponding feature.

The following steps summarize our service selection algorithm as an alternative to using case-based reasoning:

1) For all the deployment entities of the application:

a. Classify the deployment entity in one of the clusters using the clustering model
b. Find the most suitable service for the deployment entity in that cluster with respect to the entity's requirements, service price and performance

2) For each "distinct" provider that is suggested for the deployment entities (results of step 1), redo step 1 for potential services on that provider.
3) Rank potential providers based on the total price, average performance and average similarity of the services to the deployment entities

For example if in step 1 the suggested services for the deployment entities are on "Amazon" and "eApps" then in step 2, we find the suitable services for all deployment entities on "Amazon" and on "eApps". The two sets of services on these providers are then compared based on the overall price, performance, and similarity.

In order to recommend a suitable provider for the application when there is no similar application in the case base, we first search for the most suitable service for each deployment entity separately (step 1-b). The similarity is calculated based on the matches between requirements and services' features, the performance of the services for the application category (e.g. "CPU-Optimized") and price. Price is always an important factor in making decisions. The importance of the service price attribute to the customer is represented as a weight for this feature by the customer. In addition to comparing available services based on the application's and deployment entities' requirements, services are also compared based on performance. The performance information on services can be obtained by an independent third-party service like CloudHarmony [22] or the service proposed by Acs et al. [23].

The objective performance measures for most IaaS services are available in CloudHarmony. It provides independent and objective analysis on cloud services using various benchmarks to compare cloud providers. Acs et al. [23] use a hierarchical fuzzy system to reduce the complexity of the performance comparison and provide a comparable and readable performance analysis of IaaS providers. The performance objective can be based on individual resources such as CPU, memory, and disk or the overall service performance.

Cloud providers provision VMs in different categories with various configurations in terms of CPU, storage, memory and networking capacity. These categories are optimized to offer better performance with specific applications (such as computation-intensive or memory-intensive).

The most suitable service is selected based on the similarity of the service to the system requirements of the deployment entity, the service price, and the performance of the service based on the deployment entity's "category". For example, if the "category" of a deployment entity is of type "CPU-Optimized" the service price and the CPU-relevant performance of potential services are used for comparison.

A service with the highest similarity, lowest price and highest performance is the most suitable service for the deployment entity. Since this combination is not always possible for a service, the similarity, price and performance of the potential services are compared to the maximum and minimum values for these parameters amongst the top n most similar services to the deployment entity. We incorporate the customer preferences on each of these parameters by adding appropriate weights to each of these parameters.

With these conflicting criteria for selecting suitable services for deployment entities, we can use multi-criteria decision making (MCDM) approaches [24] to solve service selection problem. MCDM (a.k.a., multi-criteria decision analysis (MCDA)) is a sub-discipline of operations research, which aims to design mathematical and computational tools for selecting the best alternative among several choices with respect to several criteria [12]. Rehman et al. [25] provide a comparative study on different methods of MCDM for IaaS service selection based on performance measurements made by CloudHarmoney [22]. While their study shows MCDM techniques are effective and can be used for IaaS service selection, it also reveals that TOPSIS (Technique for Order of Preferences by Similarity to Ideal Solution) is the most suitable method for service selection when the number of available services is large. Based on their findings, we use the TOPSIS method for ranking and selecting the most suitable service for a deployment entity.

TOPSIS was proposed by Hwang and Yoon in 1981 [24]. The main idea of this method is to select an alternative that is the closest to the positive ideal solution and simultaneously the farthest from the negative ideal solution (anti-ideal solution). The distance of alternatives from positive and negative ideal solutions are calculated based on Euclidean distance [24]. The optimal solution should have the shortest distance from the ideal solution and the farthest from the anti-ideal one [12].

The procedures of the TOPSIS method in our IaaS service selection are described as follows.

Given a set of available services, $A = \{A_k \mid k = 1, \ldots, n\}$, a set of criteria (i.e., similarity, price and performance in our service selection), $C = \{C_j \mid j = 1,\ldots,m\}$, a set of values for each criterion, $X = \{x_{kj} \mid k = 1,\ldots, n; j = 1,\ldots,m\}$, and a set of weights, $W = \{w_j \mid j = 1, \ldots,m\}$, the information Table 1 = (A, C, X, W) can represented as in Table 2.

Table 1 Information table of topsis

Available services	Criteria		
	Similarity	Price	Performance
A_1	x_{11}	x_{12}	x_{13}
A_2	x_{21}	x_{22}	x_{23}
.	.	.	.
.	.	.	.
A_n	x_{n1}	x_{n2}	x_{n3}
W	α	β	γ

The first step is to calculate the normalized ratings using Eq. 2.

$$r_{kj}(x) = \frac{x_{kj}}{\sqrt{\sum_{k=1}^{n} x_{kj}^2}}, k = 1, ..., n; j = 1, ..., m. \quad (2)$$

Then the weighted normalized ratings are calculated as follows:

$$v_{kj}(x) = w_j r_{kj}(x), k = 1, ..., n; j = 1, ..., m. \quad (3)$$

The next step is to find the positive (PIS) and negative (NIS) ideal solutions. These solutions are derived as:

$$PIS = \left\{v_1^+(x), v_2^+(x), ..., v_j^+(x), ..., v_m^+(x)\right\} = \left\{\left(\ max_k v_{kj}(x) \middle| j \in J_1\right), \left(\ min_k v_{kj}(x) \middle| j \in J_2\right) \mid k = 1, ..., n\right\}, \quad (4)$$

$$NIS = \left\{v_1^-(x), v_2^-(x), ..., v_j^-(x), ..., v_m^-(x)\right\} = \left\{\left(\ min_k v_{kj}(x) \middle| j \in J_1\right), \left(\ max_k v_{kj}(x) \middle| j \in J_2\right) \mid k = 1, ..., n\right\}, \quad (5)$$

where J_1 are the benefit attributers (larger is better, e.g., similarity and performance) and J_2 are cost attributes (smaller is better, e.g., price).

The next step is to calculate the distance of each available service from the positive and negative ideal solutions (i.e., D_h and D_l respectively) using Euclidean distance,

Table 2 Similarity, price and performance of 5 most similar services to a deployment entity

Service number	Similarity	Price	Performance
1	0.95	1.18	87
2	0.94	1.10	87
3	0.94	1.08	85
4	0.90	1.00	90
5	0.98	1.28	88

$$D_h = \sqrt{\sum_{j=1}^{m} \left[v_{kj}(x) - v_j^+(x)\right]^2}, k = 1, ..., n \quad (6)$$

and

$$D_l = \sqrt{\sum_{j=1}^{m} \left[v_{kj}(x) - v_j^-(x)\right]^2}, k = 1, ..., n. \quad (7)$$

The similarity of the available services to PIS is then calculated as:

$$Similarity\ index = \frac{D_l}{D_h + D_l} \quad (8)$$

We use α, β, and γ as the weights for similarity, price and performance criteria respectively, where α, β and γ are calculated using the weights that the customer assigns for each of the requirements as follows:

$$\alpha = \frac{Avg\ (W^-)}{total\_weight}, \beta = \frac{W_{price}}{total\_weight}, \text{and } \gamma = \frac{W_{perf}}{total\_weight} \quad (9)$$

where W is the set of customer-assigned weights to the requirements of the deployment entity, $W^- \subseteq W = W - \{W_{price}\}$, W_{price} is the price weight and W_{perf} is the performance weight. This performance weight is determined by a domain expert. The *total_weight* is defined as follows.

$$\text{total\_weight} = Avg\ (W^-) + W_{price} + W_{perf} \quad (10)$$

where $Avg\ (W^-)$ is the average of all the customer-assigned weights except the price weight.

For example, assume that a deployment entity has the following customer-assigned weights to the requirements (in range [0,10]): $W_{vCPU} = 6$, $W_{Memory} = 6$, $W_{Storage} = 4$, $W_{Region} = 5$, $W_{Price} = 10$, $W_{OS} = 10$, $W_{Availability} = 6$, $W_{I/O\ Performance} = 10$, $W_{DataTransfer} = 4$. The expert set the performance weight as $W_{Perf} = 5$.

$$Avg(W^-) = \frac{6 + 6 + 4 + 5 + 10 + 6 + 10 + 4}{8} = 6.375$$

$$total\_weight = Avg(W^-) + W_{price} + W_{perf} = 6.375 + 10 + 10 = 26.375$$

$$w_{similarity} = \alpha = \frac{Avg\ (W^-)}{total\_weight} = \frac{6.375}{26.375} = 0.242$$

$$w_{price} = \beta = \frac{W_{price}}{total\_weight} = \frac{10}{26.375} = 0.379$$

$$w_{performance} = \gamma = \frac{W_{perf}}{total\_weight} = \frac{10}{26.375} = 0.379$$

Finally the available services are ranked by their similarity index and the top service is selected as the best solution.

For example, consider the services with similarities, performances and prices for a deployment entity as shown in Table 1.

Using TOPSIS and the weights (α, β and γ) above, the similarity index of the services are as follows: service1 = 0.37, service2 = 0.62, service3 = 0.65, service4 = 0.82, service5 = 0.20. The ranking of these services is: 4,3,2,1, and 5. Therefore, service 4 is selected for the deployment entity.

Studies show that deploying cloud applications on federated clouds [18], can bring several benefits including cost effectiveness, scalability, fault tolerance, and reliability [26–28]. We therefore perform an initial search for the most similar service for the entire application from the available services of all providers, given that there is no customer preference for specific providers. Although federated clouds may decrease the total price of the deployment entities, applications with multiple deployment entities may suffer from performance degradation and price increases due to communication and data transfer costs between entities deployed on different cloud providers. Inter-cloud communication and interoperability issues remain challenging for cloud providers, which makes deployment over federated clouds a less desirable option in most cases [27]. There are also other challenges related to federated clouds like security, management and monitoring [26, 28]. However, federated clouds could be a much cheaper option for applications that do not require a high amount of communications between their different deployment entities.

Given the difficulties with federated clouds, while federated clouds solutions are considered, we currently favor a list of services on the same provider for performance purposes. Comparing the total price, the average similarity and performance of the services for the whole application, we can rank potential providers and recommend the best match.

Service consolidation

Although the service search engine finds the best service (i.e., VM in our case) for each deployment entity of the application, in a typical scenario the VM is underutilized. To increase the resource utilization of selected cloud services, the service consolidator integrates as many deployment entities as possible in each service, thus decreasing the number of required services for application deployment. The final configuration must support all the requirements of the application and the preferences of the customer.

In our approach, we start with the largest service in the list of suggested services for the application. We use the price as an indicator of service capacity. Then, we accommodate as many deployment entities as possible in this service.

Next, we upgrade the service and consolidate more deployment entities in the service. If the new service configuration (i.e., the upgraded service) has an equal or lower price than the earlier configuration of all consolidated services, the upgrade is positive and acceptable. We continue the same process for the remaining deployment entities of the application. Algorithm 1 illustrates this procedure in detail.

To consolidate deployment entities in a service, we cast consolidation into the knapsack problem [29]. In this case the knapsack is the largest service that is underutilized. A greedy approximation algorithm [29, 30] is used to solve this knapsack problem.

We set the following rules on the consolidation process:

1- Two instances of the same deployment entity cannot be consolidated onto the same service to maintain the system's fault tolerance features.

2- Consolidated deployment entities must be of the same operating system.

3- The region for the consolidated deployment entities must be the same or the deployment entities must have low preference on the region (which means satisfying other requirements, e.g., price, is more important than the deployment region).

Service consolidation has advantages and disadvantages. Consolidating deployment entities with intercommunication reduces the network overhead (and cost) and increases the application's performance. However, service consolidation poses challenges related to fault tolerance. In our step - by-step case study in Summary section we demonstrate a comprehensive example of using consolidation for utility improvement.

QuARAM service recommender architecture

The architecture of QuARAM Service Recommender is illustrated in Fig. 3. It is composed of 5 main components: *requirement/deployment entity extractor*, *recommendation manager*, *case-based recommender*, *service search engine*, and *service consolidator*. It also includes three knowledge bases: *application case base*, *adaptation case base* and *providers knowledge base*. Components in dash-line boxes (i.e. the *Service search engine* and the *Service Consolidator*) may or may not participate in the process of recommendation based on the circumstances.

Algorithm 1: Service Consolidation

```
Input: List of application deployment entities and the assigned services to the entities
Output: The services and associated entities
begin
    tempServices ← services;
    tempEntities ← Entities;
    bestPrice ← CalculatePrice(Application);
    Sort(tempServices, price); // Sort services based on the services' prices
    largestservice ← tempServices(top);
    remainingServices ← tempServices-largestservice;
    remainingEntities ← tempEntities- tempEntities [largestservice ];
    while (remainingEntities ≠ ∅) do
        Calculate extra resources of largestservice;
        Sort(remainingServices, Price) ;   /* Sort services and entities based on Price of
         the services */
        for all Entity ∈ remainingEntities do          /* Merge as many entities as possible to
         largestservice */
            if Mergable(Entity, largestservice) then     /* If there is enough resources for the
             entity and the entity OS requirement is the same as the OS the service
             provides */
                Merge(Entity, largestservice);
                Remove(Entity, remainingEntities);
                Remove(service, remainingServices) ;      /*  the assigned service to the
                 entity */
                ;
            end
        end
        Update(tempServices);
        Update(tempEntities);
        bestPrice ← CalculatePrice(TemporaryApplication);
        while (price ¬ increasing ∧ upgrade possible) do
            Upgrade(largestservice);
            Calculate extra resources of largestservice;
            for all remainingEntities do        /* Merge as many entity as possible to largest
             service */
                if Mergable(Entity, largestservice) then      /* If there is enough resources for
                 Entity */
                    Merge(Entity, largestservice);
                    Remove(Entity, remainingEntities);
                    Remove(service, remainingServices);
                end
            end
            newPrice ← CalculatePrice(TemporaryApplication);
            if (newPrice≤bestPrice) then
                bestPrice ← newPrice;
                Update(tempServices);
                Update(tempEntities);
            else
                UndoUpgrade();
                UndoMerges();
            end
        end
        largestservice ← NextLargest(tempServices);
        remainingServices ← remainingServices-largestservice;
        remainingEntities ← remainingEntities- remainingEntities [largestservice ];
    end
    services ←tempServices;
    Entities ← tempEntities;
end
```

A query to the system is in the form of a TOSCA [31] Service Template for an application, which is defined using a set of *Normative Types*. This template includes the deployment entities of the application, application requirements, and customer preferences. The QuARAM Service Recommender extracts the information and sends it to the *recommendation manager*, which sends a query based on this information to the *case-based recommender* subsystem. The *case-based recommender* searches for similar cases and proposes a solution. The recommended solution is given a similarity value that indicates how similar it is to the target case. If the customer is not satisfied with the recommendations or the similarity values of all solutions are less than a specified threshold, the system searches for a suitable service for each deployment

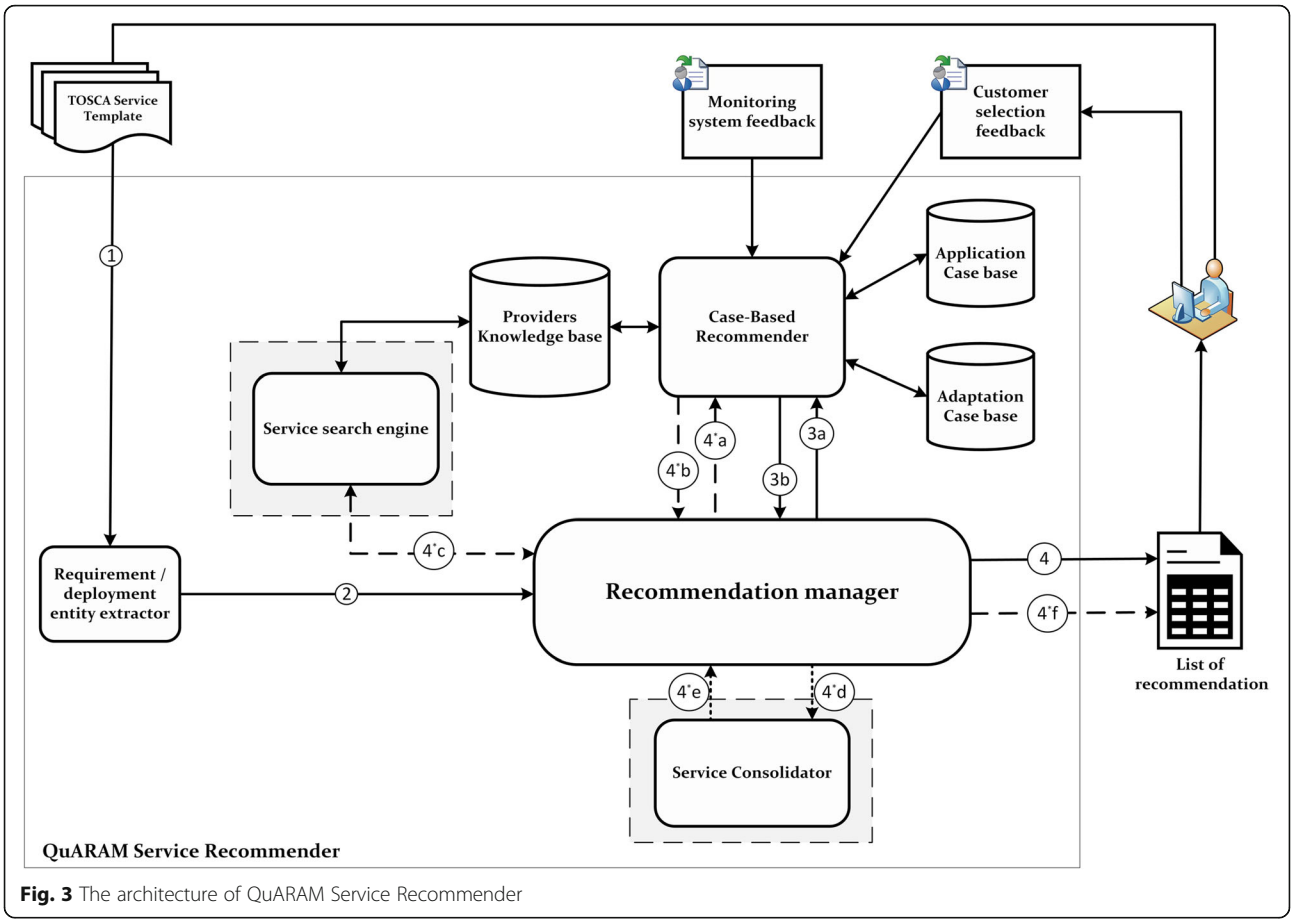

Fig. 3 The architecture of QuARAM Service Recommender

entity separately. This search involves both the *case-based recommender* and the *service search engine* which searches among all available offerings in the clouds. When the system finds a suitable service for every deployment entity, it consolidates the services, where possible, to improve the resource utilization and reduce the deployment costs. Then, a set of recommendations is suggested to the customer. Based on the selection of the customer (customer selection feedback) a case is added to the case base. Monitoring systems also provide feedback on the service performance with respect to QoS requirements. This feedback is used to update the case bases. In this section we describe the different components of the QuARAM Service Recommender system and how they interact to provide recommendations for service selection.

QuARAM service recommender components

Requirement/deployment entity extractor

A challenge in cloud service selection is specifying cloud applications in such a way that a system can automatically identify and extract the requirements and customer preferences from the specification. TOSCA [31] (Topology and Orchestration Specification of Cloud Applications) is a standard specification method for cloud applications that allow integration of the application's requirements (e.g., hardware requirements and QoS), customer preferences and characterizations of the application (e.g., application type) into the specification. With a standard specification, brokers can identify and extract the requirements and characterizations automatically, select suitable services, provision the service instances, configure and deploy the application on the cloud.

The *Requirement/deployment entity extractor* component receives the TOSCA Service Template of the application from the customer and extracts the application requirements, customer preferences, and the deployment entities of the application. It provides a couple of *.csv* documents based on this information and sends them to the *recommendation manager*, which in turn distributes them to the other components. To evaluate, select and integrate services for an application deployment, the platform uses the case-based recommender, the service search engine and the service consolidator components.

Case-based recommender

The *Case-Based recommender* receives the requirements of an application or one of its deployment entities as input, searches the case base for similar cases, and returns a list of recommended deployment configurations to the target application. The *Case--Based recommender* has access to three knowledge bases. The first one is the *application case base* which contains previously deployed applications/ deployment entities, their requirements and the suitable platform configuration for cloud deployment. The *application case base* also includes the customer preferences and the SLAs. The second knowledge base is the *adaptation case base* which incorporates the knowledge about how to adapt a solution so that it fits the features of the target problem (i.e., the new application). The third knowledge base is the *providers knowledge base*. This knowledge base contains knowledge about the available cloud service offerings, the performance of each service from different perspectives (e.g., computation, I/O, etc.) and knowledge about the transition from one service to another. The implementation and evaluation of the *Case-Based recommender* is presented in detail in Soltani et al. [2, 6]. Our experimental results show up to 90% precision of recommended services using case-based recommender.

Service search engine

The *service search engine* uses the approach described in Hybrid service selection section to search the available offerings from cloud providers for a suitable service for the application's deployment entities, based on the requirements of each entity, customer preferences and the performance of cloud services. It uses TOPSIS to rank potential services and returns a list of ranked suitable services for the application's deployment entities.

Service consolidator

This component consolidates the services proposed for a set of deployment entities, using the algorithm presented in Service consolidation section, to recommend a list of suitable configurations for the application deployment on the cloud. The final configuration must support all the requirements of the application and the preferences of the customer. The *service consolidator* takes into consideration the preferences of the customer with respect to the service price and performance. This component uses the *greedy approximation algorithm* to handle the problem of large search space for consolidating multiple deployment entities into services.

Recommendation manager

The *Recommendation Manager* is the core component of the QuARAM Service Recommender that manages and coordinates the various components. The *recommendation manager* receives the requirements of the customer's application and its deployment entities from the *requirement/deployment entity extractor*. Then, it sends these requirements to the *case-based recommender,* which returns a list of recommendations, along with their similarity to the customer's query and the adapted solutions. Based on the similarity of the retrieved cases, the *recommendation manager* decides whether to send the recommendations to the customer based on a pre-specified similarity threshold. The top 5 recommendations (or the top n that have similarity above the threshold where $n <= 5$) are sent to the customer to choose from if all of the top 5 have similarity above the threshold. If none of the retrieved cases scores has a similarity above the threshold, the *recommendation manager* uses the information of the deployment entities of the application to find a more fine-grained configuration for the application deployment. All proposed configurations are sent to *service consolidator*, which returns a list of aggregate recommendations that best fit the whole application. The recommendation manager then sends this list to the customer.

Feedback components

Customer selection feedback and monitoring system feedback are responsible to provide the feedback from customer and monitoring system to the case-based recommender for maintenance and system improvement.

Summary of recommendation process

The following steps summarize the recommendation process (as illustrated in Fig. 3):

1) The customer sends the Service Template of his/her application to the QuARAM Service Recommender.
2) The r*equirements/deployment entity extractor* parses the template and extracts a list of the application's deployments entities, requirements and customer preferences and sends them to the *recommendation manager*.
3) The *recommendation manager* creates a query based on the application requirements, the deployment entities and customer preferences and sends it to the *case-based recommender (a).* The *case-based recommender* provides a list of recommendations for the application deployment using the knowledge bases (application case base, adaptation case base, and providers knowledge base). The list of recommendations along with the

case similarity is then sent back to the *recommendation manager (b).*

The *recommendation manager* decides on the next step based on the similarity of retrieved cases and a specified similarity threshold.

4) The *recommendation manager* sends to the customer a list of retrieved cases whose similarity is above the threshold. In the context of the QuARAM framework the list is sent to the *Deployment engine* to pass it to the customer.

4*) If none of the retrieved cases has a similarity above the specified threshold then, for each of the application's deployment entities the *recommendation manager* performs the following steps:

a. Query the *case-based recommender* for suitable services for the entity.
b. The *case-based recommender* returns a list of recommendations for the deployment entity to the *recommendation manager.*
c. If the similarity of all retrieved cases is below the specified threshold then *recommendation manager* queries the *service search engine* and receives a list of recommendations for the deployment entity. This list could be empty which means there is no available service that fits the entity requirements.
d. The *recommendation manager* sends all the application's deployment entities information and the recommended services to the *service consolidator.*
e. The *service consolidator* returns a more cost-efficient configuration for the deployment of the application to the *recommendation manager.*
f. The configurations are sent back to the customer (or the *deployment engine* in the terminology of QuARAM framework.

This process assumes that the application can be deployed on federated clouds. To avoid the current challenges of federated clouds, we further perform the following steps to restrict our recommendations to only a single provider for the entire application.

1) The *recommendation manager* sends the list of deployment entities of the application and the recommended provider for each entity to the *service search engine*.
2) The *service search engine* returns a list of recommendations for the deployment of the application (i.e., all the deployment entities) on each of the providers that are listed by the *recommendation manager.*
3) The *recommendation manager* sends these recommendations (i.e., the deployment entities and the set of services recommended on each provider) to the *service consolidator.*
4) The *service consolidator* makes possible aggregation for services and sends a more cost-efficient configuration of services on each candidate provider.
5) The list is then presented to the customer to make a final selection.

Validation of the QuARAM service recommender

To validate our framework, we developed a prototype of the service recommender by extending the case-based recommender in Soltani et al. [2]. In the original case-based recommender system, the case base includes a problem part and the solution part. The problem part has the application requirements and the customer preferences as the weights for each the requirements, and the solution part describes the selected service and the configurations. In our original system, we assume that the applications requirements can be satisfied by just one VM. Here, we assume that an application may require more than one VM.

The schema for the updated application case base is given in Fig. 4. The new application case base includes cases with the problem part attributes as follows: application type (type), application tiers (tiers), maximum number of concurrent users (maximumusers), region, response time, security, availability, maximum latency (latency), number of load balancers (loadbalancers), number of servers, servers, priority (weight for each of the attributes), and price. Some of the features are using just for comparison of the previously deployed applications and the target application in the case base (e.g. maximumusers).

The "Number of servers" attribute contains the number of deployment entities used by the application. The "servers" attribute is an array of attributes for each of the servers in the application. The attributes of a "server" are as follows:

• Memory	• NOSQL storage (NoSQL)
• CPU power	• application server (AppServer)
• number of CPU cores (VCPU)	• storage (Amount of required storage for the application)
• OS (Lonux, Windows)	• bandwidth (download, upload)
• DBMS (e.g. DB2, SQL Server, Oracle)	• priority

The solution part of a case in the case base is also updated to contain the following attributes: VMs and configuration. The "VMs" attribute represents the service instances (e.g., "*m1.large, Amazon*", "*Performance1, GoGrid*"). The ServiceNumber represents the service that is used for the application and the ServiceID represents the

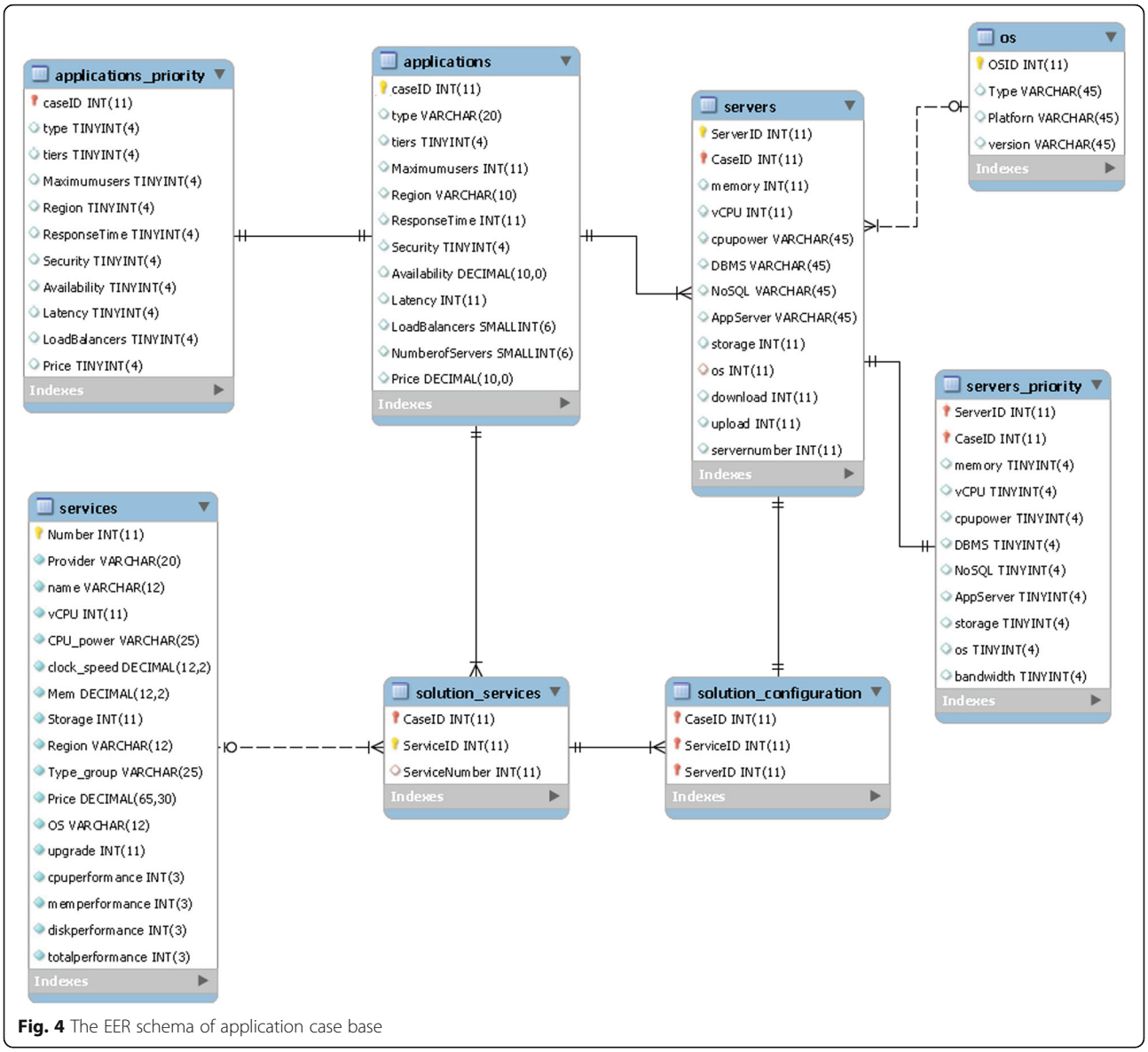

Fig. 4 The EER schema of application case base

unique instance of that service for the deployment of the application. The "configuration" attribute is an array of the servers and the VMs (services) on which they are deployed. Each server and each VM is identified by a unique number (e.g., "1,1", "2,1", "3,2" which means that servers 1 and 2 are deployed on VM 1 and server 3 is deployed on VM 2). The "*Services*" table is our provider knowledge base. It contains the available services from different providers and their advertised features.

We validate the prototype with the sample application whose requirements are given in Table 3. The application includes 7 servers with two instances of one of the servers. The "0" value in any column indicates that the deployment entity has no specific requirement for this feature. We set the region to "US" for our entire experimental setups. This means that the customer prefers to deploy the application on one of the "US" regions (e.g. "US-West", "US-East", etc.). Some applications run multiple instances of the same deployment entity.

While the customer may specify the type of application in the specification, it's not necessarily that this type applies to the deployment entities. For example, an application may be of type "CPU-intensive" but it does not mean that all its deployment entities are "CPU-intensive". It may have a database server deployment entity which is "memory-intensive". To compare the performance of potential services, we need to identify the "category" of deployment entities.

Table 3 System requirements of tested application and the assigned weight for the requirement

Deployment entity	vCPU/w	Mem/w	Storage/W	Category	Price	OS
1	4/0.6	6/0.6	0/0	'CPU Purpose'	0	Win
2	2/0.6	8/0.6	0/0	'General Purpose'	0	Win
3	2/0.6	1/0.6	20/1	'General Purpose'	0	Win
3	2/0.6	1/0.6	20/1	'General Purpose'	0	Win
4	0/0	0/0	111/1	'General Purpose'	0	Lin
5	2/0.6	3/0.6	0/0	'General Purpose'	0	Win
6	8/0.8	32/0.8	50/0.6	'Memory optimized'	0	Win
7	4/0.8	8/0.8	0/0	'CPU optimized'	0	Win

Most cloud providers classify their services into the following "categories" based on VM configurations: "Compute optimized", "Memory optimized", "Storage optimized" and "General purpose". Using the configuration information (i.e., memory, CPU, and storage) of potential services, we train a classifier and use it to categorize the application deployment entities based on their system requirements. We use the WEKA data mining tool [32] and c4.5 algorithm [33] (implemented with the name J48 in WEKA) to train our classifier model. Each column has the feature and the assigned weight for that feature. Assigned weights range from 0 to 1, where 1 means "most important" and 0 is "least important".

Figure 5 shows the process flow of the recommendation system and the interactions between the various components. The *Service Template* of the application is sent to the *requirement/deployment entity extractor* (1). The deployment entities of the application, the

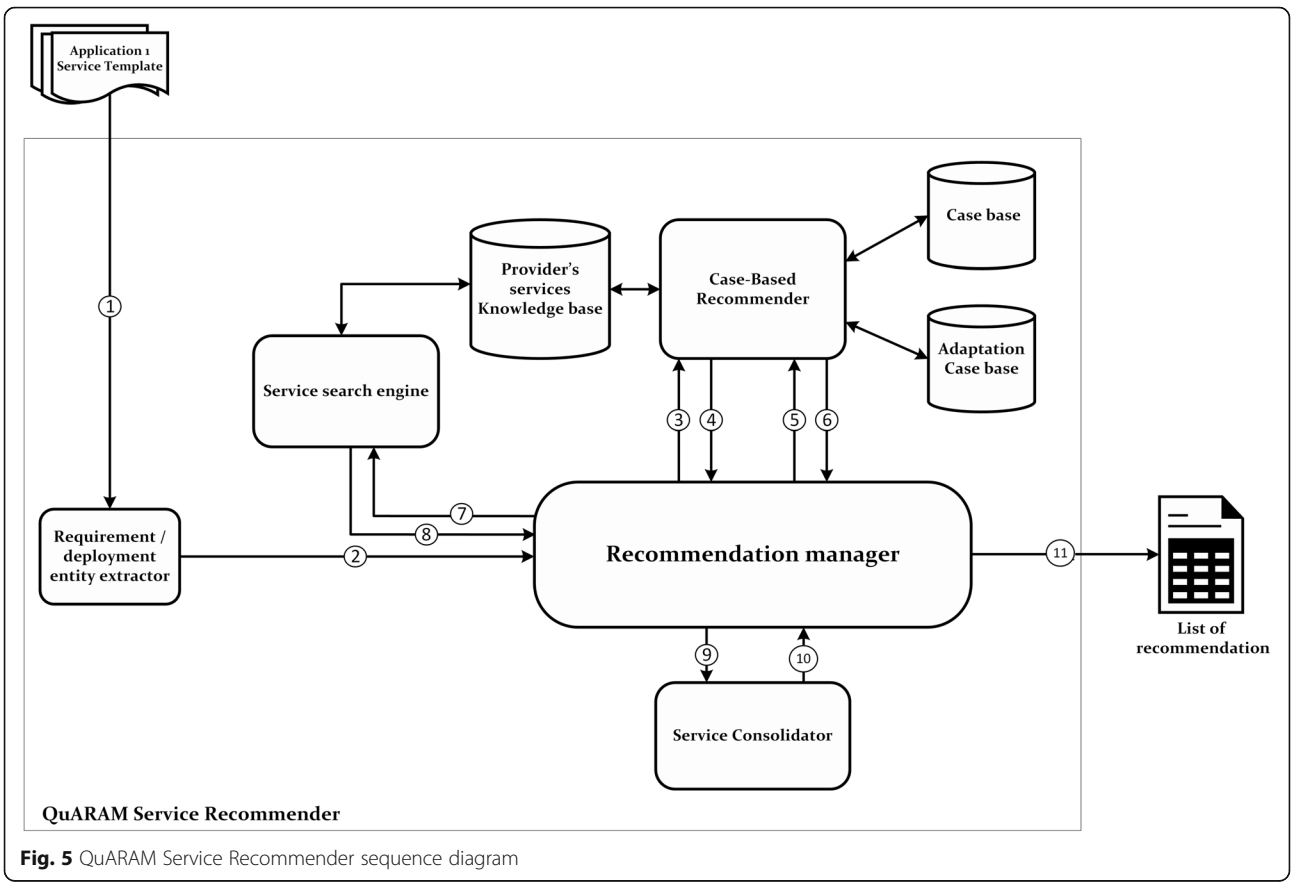

Fig. 5 QuARAM Service Recommender sequence diagram

requirements and the customer preferences are returned in the form of two *.csv* files, *application.csv* and *servers.csv*, shown in Fig. 5. *Application.csv* contains the requirements and customer preferences of the application. *Servers.csv* includes the requirements and the customer preferences of each deployment entity.

The *Recommendation manager* receives these *.csv* files (2) and queries the *case-based recommender* (3). The *case-based recommender* returns the recommendations in the form of a *.csv* file (4), which contains the servers and their candidate services (the "configuration" part of the solution).

We assume that our case base contains the applications previously deployed on cloud that have just one server which was deployed on one service.

Since in this test case there is no other application in the case base that has more than one server, the *case-based recommender* returns an empty list to the *recommendation manager* for the application.

The *recommendation manager* then creates queries for each of the deployment entities separately. Figure 6 is an example of the *.csv* files for one of the deployments entities that are passed to the *case-based recommender*.

application .CSV

```
Provider,AppType,Apptier,MaxUser,Region,Resp
onseTime,Security,Availability,Latency,LoadB
alancer,Price,RegionWeight,ResponseTimeWeigh
t,SecurityWeight,AvailabilityWeight,LatencyW
eight,LoadBalancerWeight,PriceWeight
,Compute-
intensive,3,2000,US,,8,99,,,,0.5,,0.5,0.8,,,
1
```

servers .CSV

```
Number,vCPU,CPUpower,Mem,Storage,OSType,OSPl
atform,OSVersion,DBMS,NoSQL,AppServer,Downla
od,Upload,vCPUWeight,CPUpowerWeight,MemWeigh
t,StorageWeight,OSTypeWeight,OSPlatformWeigh
t,OSVersionWeight,DBMSWeight,NoSQLWeight,App
ServerWeight,DownlaodWeight,UploadWeight
1,4,,6,,Windows,64,Server2008,SQLServer2008,
,,,,0.6,,0.6,,1,1,1,1,,,,
```

Fig. 6 The .csv files pertaining to server 1 of the application

Some values are set to null for attributes such as application tiers since they cannot map to a single server.

The queries are sent to the *case-based recommender* (5) and a list of recommendations is returned to the *recommendation manager* (6). The recommendation manager uses the solution of the most similar retrieved case as the recommended solution for the deployment entity. The recommended solution from the case-based recommender for the deployment entity in Fig. 6 is "*Amazon, m2xlarge, sim= 52.8%*", where *sim* is the similarity of the retrieved case to the entity. The threshold for accepted similarity is set as 80%. Therefore, the recommended solution with the similarity of 52.8% is rejected.

The *recommendation manager* then generates a query to the *service search engine* as illustrated in Fig. 7 (7). The search engine proposes "*Athlantic.net, Xlarge*" and returns it to the *recommendation manager* (8). The steps 5–8 are repeated for each deployment entity. Table 4 illustrates the recommended solution for each of deployment entities of our test application. The column "Recommended by (R/B)" indicates whether the solution is provided by the *case-based recommender* (CBR) or the *service search engine* (SSE). "-" in the name column indicates that the provider doesn't specify a name for the service.

Then r*ecommendation manager* sends the list of recommended services for the deployment entities to the *service consolidator* to integrate potential services (9). Table 5 shows the results of the consolidation that are returned to the *recommendation manager* (10). The total price for the application dropped to $1.98/h (compared to the non-consolidated services in Table 4) for the deployment of the application.

In this example, we assume that federated cloud is a viable option for the application deployment. If the customer prefers the deployment on a single cloud provider, the *recommendation manager* forwards the requirements of the deployment entities and the list of potential providers (i.e., *Microsoft Azure*, *eApps*, *Joyent*, and *Amazon* for this application) to the *service*

```
Number,Provider,name,vCPU,'CPU
power',Mem,Storage,Region,category,Price,OS,vCP
UWeight,MemWeight,StorageWeight,RegionWeight,Pr
iceWeight,OSWeight
1,,,4,,6,0,US,General
Purpose,0,Windows,0.6,0.6,,0.5,1,1
```

Fig. 7 Query for service search engine

Table 4 Recommended services for each deployment entity

Server	Similarity	Provider	Name	Mem (GB)	Storage (GB)	Region	Price($/hr)	OS	Storage price ($/hr)	R/B
1	86.59	eApps	–	8	75	US-West	0.182	Win	0	SSE
2	86.98	Amazon	r3.large	15.25	32	US-West	0.3	Win	0	CBR
3	86.55	Microsoft Azure	Medium(A2)	3.5	60	US-West	0.154	Win	0	SSE
3	86.55	Microsoft Azure	Medium(A2)	3.5	60	US-West	0.154	Win	0	SSE
4	82.12	eApps	1024	1	15	US-East	0.029	Lin	.0039	CBR
5	84.81	Amazon	t2.medium	4	0	US-West	0.072	Win	0	CBR
6	85.88	Joyent	'High Storage	32	7680	Us-South	0.923	Win	0	SSE
7	88.47	eApps	–	8	75	US-West	0.182	Win	0	SSE
Total Price = $2.00/h										

search engine to find the best service for each entity on each provider. The results are sent to the *service consolidator* and then to the *recommendation manager* (11) (Tables 6, 7 and 8).

Following the process of service selection step-by-step for the case study using our proposed platform shows the feasibility of our approach in IaaS service selection.

Summary

The motivation of our research comes from the need to develop a platform for cloud application deployment to cope with the growing cloud market. Although the market growth provides economic benefits to customers due to increased competition between providers, the lack of similarity with respect to how services are described and priced by different providers makes the decision on the best option challenging. The decision also needs to consider the customer's preferences over different features. We recognized three challenges of automatic service selection: 1) Automatic extraction of application requirements and customer preferences, 2) Selection of suitable services from a large pool of available services that is constantly growing. The heterogeneity and the large number of selection criteria pose additional challenges. Consolidation of selected services to maximize the resource utilization, minimize the deployment price, and improve the application performance is required, 3) Adaptation to the dynamic cloud environment.

The 1st and 3rd challenges are addressed in our previous works [2, 6]. In this paper, we focus more on 2nd challenge. We adopted our case-based recommender that was presented in Soltani et al. [2] and extended it in a number of ways to improve the precision of the recommendations. We incorporate an alternate method of service recommendation using MCDM when the case base lacks similar previous application deployments. Consolidation of the services is further introduced in the process to improve resource utilization and lower deployment costs where applicable. A step-by-step case study of the recommendation process using our prototype shows the feasibility of the proposed methods and techniques in service recommendation for application deployment on cloud.

Our platform adopts a hybrid approach in service selection, where we alternate between case-based reasoning and MCDM based on the presence of similar cases in the case base. The solutions generated with MCDM are currently not automatically added to the case base in order to extend its coverage for future

Table 5 Results of service consolidation for the tested application

Service	Provider	Price	OS	Servers	Additional disk size	Total price
1	Amazon	0.3	Win	2	0	0.3
2	Azure	0.154	Win	3	0	0.154
3	Azure	0.154	Win	3	0	0.154
4	eApps	0.029	Lin	4	96	0.029
5	Amazon	0.072	Win	5	0	0.072
6	Joyent	0.923	Win	6	0	0.923
7	eApps	0.349	Win	1,7	0	0.349

Table 6 Results of service consolidation for the tested application on microsoft azure

Service	Price	OS	Servers	Additional disk size	Total price
0	0.592	Win	1,3,5	0	0.592
1	0.31	Win	2	0	0.31
3	0.154	Win	3	0	0.154
4	0.02	Lin	4	91	0.026233
6	1.2	Win	6	0	1.2
7	0.6	Win	7	0	0.6

Table 7 Results of service consolidation for the tested application on joyent

Service	Price	OS	Servers	Additional disk size	Total price
0	1.02	Win	1,2,3	0	1.02
4	0.02	Lin	4	91	0.02536
6	0.923	Win	6	0	0.923
7	1.02	Win	3,7	0	1.02

deployments. We plan to develop techniques that can better update the information in case bases for use in future searches. An offline simulator could perform this task (i.e., re-select services) for already deployed applications and provide the information to retain in the case base when new services or providers are registered into the system.

Based on the studies on service selection using MCDM, we use TOPSIS [24] for the search-based service selection to combine with case-based reasoning. We are interested in studying the performance of other MCDM methods [24] (e.g., ELECTRE,[2] PROMETHEE,[3] etc.) and hybrid methods [34] (e.g., combination of EMO[4] and MCDM methods) compared with TOPSIS.

There are several limitations in the proposed work which require further study. The deployment of an application's component as independent deployment entities entails communications between these entities. This communication presents data transfer overhead and incurs unnecessary network traffic. The current platform ignores the network overhead in decision making. We plan to study the effect of this deployment method on both the network load and application performance. We also plan to quantitatively study how service consolidation may alleviate the overhead on the network and lower communication costs. Our focus in this paper is on the selection of IaaS service model using case-based reasoning and MCDM. We plan to check the feasibility of the proposed approach for SaaS and PaaS service models and compare the results with the state of the art methods for these two cloud service models.

Table 8 Results of service consolidation for the tested application on amazon

Service	Price	OS	Servers	Additional disk size	Total price
0	0.752	Win	1,2,3	0	0.752
3	0.1	Win	3	20	0.10274
4	0.026	Lin	4	111	0.041205
5	0.1	Win	5	0	0.1
6	1.08	Win	6	0	1.08
7	0.6	Win	7	0	0.6

Endnotes

[1]Defined by the customer as the importance of the requirement

[2]Elimination Et Choix Traduisant He Realite

[3]Preference Ranking Organization Method for Enrichment Evaluations

[4]Evolutionary Multi-Objective Optimization

Abbreviations

AHP: Analytical Hierarchy Process; ANP: Analytic Network Process; BSC: Balanced Scorecard; CBR: Case-Based Reasoning; CP: Constraint Programming; EER: Enhanced Entity Relationship; ELECTRE: Elimination Et Choix Traduisant He Realite; EMO: Evolutionary Multi-Objective Optimization; FDM: Fuzzy Delphi Method; IaaS: Infrastructure as a service; IRVM: Improved Ranked Voting Method; K-NN: K-Nearest Neighbor; MCDM: Multi-criteria Decision Making; PROMETHEE: Preference Ranking Organization Method for Enrichment Evaluations; QoS: Quality of Service; QuARAM: QoS-aware cloud application management; SLA: Service Level Agreement; SSE: Service Search Engine; TOPSIS: Technique for Order of Preferences by Similarity to Ideal Solution; VM: Virtual Machine

Acknowledgements

We acknowledge Natural Sciences and Engineering Research Council of Canada (NSERC) for funding this project.

Funding

This project was funded by NSERC (Natural Sciences and Engineering Research Council of Canada) as a part of PhD project.

About the Authors

Dr. Sima Soltani received her PhD in Computer Science in 2016 from Queen's University, Canada. She received her MSc. in software engineering in 2007 from Amirkabir University of Technology and her BSc in software engineering in 2004 from Isfahan University, Iran. She was faculty member of Islamic Azad University and was instructor for multiple universities in Iran. Her research interest include cloud computing, data mining and data analysis, big data and automatic computing systems.

Prof. Patrick Martin is a Professor in the School of Computing at Queen's University and the Director of the Database Systems Laboratory. He is a Faculty Fellow and Visiting Scientist with IBM's Centre for Advanced Studies, a Scotiabank Scholar, a member of the Southern Ontario Smart Computing Innovation Platform (SOSCIP) Scientific Committee and a member of the Advisory Panel on Analytics for the Ontario Brain Institute. His research interests include big data analytics, database system performance, cloud computing and autonomic computing systems.

Dr. Khalid Elgazzar is an assistant professor with the Center of Advanced Computer Studies at the University of Louisiana at Lafayette. Before joining CACS, he was an NSERC postdoctoral fellow at Carnegie Mellon School of Computer Science. He received his PhD in Computer Science in 2013 from the School of Computing at Queen's University in Canada. Dr. Elgazzar received the prestigious NSERC PDF award (2015–2017) from the Canadian Government and the 2014 distinguished re-search award from Queen's University. He also received sever-al recognition and best paper award at top international venues. His research interests span the areas of mobile and ubiquitous computing, context-aware systems, Internet of Things, and elastic networking paradigm. Dr. Elgazzar is currently leading a team to create the next generation of IoT open stack, get-ting every connected device onboard to bring the future even closer.

Authors' contributions

PM and KE were PhD supervisor and co-supervisor of SS for this project. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]School of Computing, Queen's University, Kingston, Canada. [2]School of Computing and Informatics, University of Louisiana, Lafayette LA70503, USA.

Received: 15 January 2018 Accepted: 28 June 2018
Published online: 11 July 2018

References

1. Buyya R, Yeo CS, Venugopal S (2008) Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In: 2008 10th IEEE international conference on high performance computing and communications. IEEE, Dalian, pp 5–13
2. Soltani S, Martin P, Elgazzar K (2014) QuARAMRecommender: case-based reasoning for IaaS service selection. In: The international conference on cloud and autonomic computing (CAC 2014). IEEE, London, pp 220–226
3. Best Infrastructure as a Service(IaaS Providers. https://www.g2crowd.com/categories/infrastructure-as-a-service-iaas?order=alphabetical&page=3#product-list. Accessed 5 Jan 2018
4. Cloudorado. https://www.cloudorado.com/. Accessed 5 Jan 2018
5. Zhang M, Ranjan R, Haller A, et al (2012) Investigating decision support techniques for automating Cloud service selection. 4th IEEE Int Conf Cloud Comput Technol Sci Proc 759–764. https://doi.org/10.1109/CloudCom.2012.6427501
6. Soltani S, Elgazzar K, Martin P (2016) QuARAM service recommender: a platform for IaaS service selection. In: 9th international conference on utility and cloud computing. ACM Press, Shanghai, pp 422–425
7. Martin P, Soltani S, Powley W, Hassannezhad M (2013) QoS-aware cloud application management. In: Catlett C, Gentzsch W, Grandinetti L et al (eds) Cloud computing and big data. IOS Press, Amsterdam
8. Han S, Hassan MM, Yoon C, Huh E (2009) Efficient Service Recommendation System for Cloud. In: Proceeding: International Conference on Grid and Distributed Computing, GDC 2009, Held as Part of the Future Generation Information Technology Conferences, FGIT 2009. Jeju Island, pp 117–124
9. Zeng W, Zhao Y, Zeng J (2009) Cloud service and service selection algorithm research. In: The first ACM/SIGEVO summit on genetic and evolutionary computation - GEC 09. ACM Press, New York, pp 1045–1048
10. Rehman ZU, Hussain FK, Hussain OK (2011) Towards multi-criteria cloud service selection. In: 2011 fifth international conference on innovative mobile and internet Services in Ubiquitous Computing. Ieee, Seoul, pp 44–48
11. Chen C, Yan S, Zhao G, et al (2012) A systematic framework enabling automatic conflict detection and explanation in cloud service selection for enterprises. In: 2012 IEEE fifth international conference on cloud computing. IEEE, Hawaii, pp 883–890
12. Whaiduzzaman M, Gani A, Anuar NB et al (2014) Cloud service selection using multicriteria decision analysis. Sci World J 2014:1–10. https://doi.org/10.1155/2014/459375
13. Godse M, Mulik S (2009) An Approach for Selecting Software-as-a-Service (SaaS) Product. In: 2009 IEEE International Conference on Cloud Computing. IEEE, Bangalore, pp 155–158
14. Do CB, Kwang-Kyu S (2015) A cloud service selection model based on analytic network process. Indian J Sci Technol 8:1–5. https://doi.org/10.17485/ijst/2015/v8i18/77721
15. Zhang M, Ranjan R, Menzel M, Haller A (2012) A declarative recommender system for cloud infrastructure services selection. In: 9th international conference on economics of grids, clouds, systems, and services, GECON 2012. Springer, Berlin and Heideiberg, pp 102–113
16. Qian H, Zu H, Cao C, Wang Q (2013) CSS: facilitate the cloud service selection in IaaS platforms. In: 2013 International Conference on Collaboration Technologies and Systems (CTS). IEEE, San Diego, pp 347–354
17. Lee S, Seo K (2016) A hybrid multi-criteria decision-making model for a cloud service selection problem using BSC, fuzzy delphi method and fuzzy AHP. Wirel Pers Commun 86:57–75. https://doi.org/10.1007/s11277-015-2976-z
18. Baranwal G, Prakash Vidyarthi D (2016) A cloud service selection model using improved ranked voting method. Concurr Comput Pract Exp 28:3540–3567. https://doi.org/10.1002/cpe.3740
19. Sundareswaran S, Squicciarini A, Lin D (2012) A Brokerage-Based Approach for Cloud Service Selection. In: 2012 IEEE Fifth International Conference on Cloud Computing. IEEE, Hawaii, pp 558–565
20. Han J, Kamber M (2006) Data mining: concepts and techniques. Morgan Kaufmann, p 800
21. Duda RO, Hart PE, Stork DG (2001) Pattern Classification. Wiley- Interscience Publication, Canada, p 280
22. CloudHarmony transparency for cloud. http://cloudharmony.com. Accessed 5 Jan 2018
23. Acs S, Zsolt N, Gergely M (2014) A Novel Approach for Performance Characterization of IaaS Clouds. In: Euro-Par 2014: Parallel Processing Workshops. Springer International Publishing, Porto, pp 109–120
24. Tzeng G-H, Huang J-J (2011) Multiple attribute decision making, Methods and Applications. Boca Raton, CRC Press
25. Rehman ZU, Hussain OK, Hussain FK (2012) Iaas Cloud Selection using MCDM Methods. 2012 IEEE Ninth Int Conf E-bus Eng 246–251. https://doi.org/10.1109/ICEBE.2012.47
26. Subramanian T, Savarimuthu N (2015) A Study on Optimized Resource Provisioning in Federated Cloud. CoRR abs/1503.03579, p 1–6
27. Toosi AN, Calheiros RN, Buyya R (2014) Interconnected cloud computing environments. ACM Comput Surv 47:1–47. https://doi.org/10.1145/2593512
28. Shareef O, Kayed A (2015) A survey on federated clouds environment. Int J Adv Res Comput Sci Softw Eng 5:83–92
29. Neapolitan RE, Naimipour K (2011) Foundation of algorithms, 4 edn. Jones Bartlett Learn, Sudbury. http://common.books24x7.com.proxy.queensu.ca/toc.aspx?bookid=35300. Accessed 10 Sept 2015
30. Williamson DP, Shmoys DB (2011) The Design of Approximation Algorithms. Cambridge University Press, Cambridge
31. OASIS. Topology and orchestration specification for cloud applications (TOSCA) Version 1. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. Accessed 12 Dec 2014
32. Weka: Datamining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/. Accessed 2 June 2015
33. Mitchell TM (1997) Machine learning. McGrawHill, New York
34. Odu GO, Charles-Owaba OE (2013) Review of multi-criteria optimization methods – theory and applications. IOSR J Eng 3:01–14. https://doi.org/10.9790/3021-031020114