Compatibility-Aware Cloud Service Composition under Fuzzy Preferences of Users

Amir Vahid Dastjerdi, *Member, IEEE* and Rajkumar Buyya, *Senior Member, IEEE*

Abstract—When a single Cloud service (i.e., a software image and a virtual machine), on its own, cannot satisfy all the user requirements, a composition of Cloud services is required. Cloud service composition, which includes several tasks such as discovery, compatibility checking, selection, and deployment, is a complex process and users find it difficult to select the best one among the hundreds, if not thousands, of possible compositions available. Service composition in Cloud raises even new challenges caused by diversity of users with different expertise requiring their applications to be deployed across difference geographical locations with distinct legal constraints. The main difficulty lies in selecting a combination of virtual appliances (software images) and infrastructure services that are compatible and satisfy a user with vague preferences. Therefore, we present a framework and algorithms which simplify Cloud service composition for unskilled users. We develop an ontology-based approach to analyze Cloud service compatibility by applying reasoning on the expert knowledge. In addition, to minimize effort of users in expressing their preferences, we apply combination of evolutionary algorithms and fuzzy logic for composition optimization. This lets users express their needs in linguistics terms which brings a great comfort to them compared to systems that force users to assign exact weights for all preferences.

Index Terms—Cloud computing, Cloud service composition, ontology, service level agreement, quality of service

1 Introduction

In order to deliver their solutions, application service providers can either utilize the platform-as-a-service (PaaS) offerings such as Google App Engine and OpenShift or develop their own hosting environments by leasing virtual machines from infrastructure-as-a-service (IaaS) providers like Amazon EC2 or GoGrid. However, most PaaS services have restrictions on the programming language, development platform, and databases that can be used to develop applications. Such restrictions encourage service providers to build their own platforms using IaaS service offerings.

One of the key challenges in building a platform for deploying applications is to automatically compose, configure, and deploy the necessary application that consists of a number of different components. If we consider the deployment requirements of a web application service provider, it will include security devices (e.g., firewall), load balancers, web servers, application servers, database servers, and storage devices. Setting up such a complex combination of appliances is costly and error prone even in traditional hosting environments [1], let alone in Clouds. Virtual appliances provide an elegant solution for this problem. They are built and configured with a necessary operating system and software packages to meet software requirements of a user.

In IBM smart Cloud, Amazon EC2, GoGrid, Rackspace, and other key players in the IaaS market, users first have to select their software solution (asset catalogs, images, etc.) and then select the proper virtual machine configuration (e.g., instance type) to host the software solution. Furthermore, a user may require more than one virtual appliance and machine, and a composition of them that can meet all the requirements of users is required. However, the selection of the best composition is a complex task and none of the providers provide any ranking system to choose the best instance type and software solution for the deployment.

In addition, the best choices found for individual appliances cannot be simply put together as they may not be compatible with the hosting environment. For example, if an appliance format is Open Virtualization Format (OVF) it cannot currently be deployed on Amazon EC2. Moreover, there exist legal constraints imposed by countries such as the USA on importing and exporting of appliances from a provider to another. Dealing with all these complexities is costly and aggravating for unskilled users and encourages them to seek professional help.

In this paper, to simplify the process of selecting the best virtual appliance and unit (computing instance) composition, a novel framework is presented. The framework proposes:

- An approach to help non-expert users with limited or no knowledge on legal and virtual appliance image format compatibility issues to deploy their services flawlessly. For this purpose, we first automatically build a repository of Cloud services in web service modeling language (WSML) [2] and then enrich it with experts' knowledge (lawyers, software engineers, system administrators, etc.) on the aforementioned constraints. The knowledgebase then is used for reasoning in an algorithm that identifies whether a set of Cloud services consisting of virtual appliance and units are compatible or not.

• *The authors are with the Department of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3053, Australia. E-mail: amir.vahid@unimelb.edu.au, rbuyya@unimelb.edu.au.*

Manuscript received 6 Sept. 2013; revised 18 Nov. 2013; accepted 7 Jan. 2014; date of publication 15 Jan. 2014; date of current version 30 Apr. 2014.
Recommended for acceptance by A. Liang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2300855

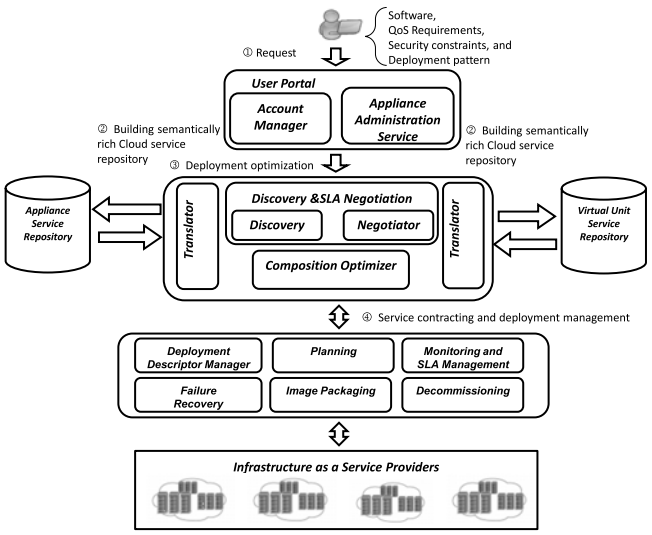


Fig. 1. Architecture's main components that facilitates QoS-aware deployment of user applications.

- A Cloud service composition optimization technique that allows non-expert Cloud users to set their preferences using high level if-then rules and get user friendly recommendations on the composition solution's prominence. The majority of end users avoid systems that incur complexity in capturing their constraints, objectives and preferences. An example of such systems is the one which require users to assign weights to their objectives. In this case, users have to find a way to prioritize their preferences and then map them to weights. After that, the system has to find out how precise users have gone through the process of weight assignment. To tackle this issue, a major objective of this research is to offer ranking system for Cloud service (i.e., virtual appliance and machine) composition that let users express their preferences conveniently using high-level linguistic rules. Our system then utilizes multi-objective evolutionary approaches and fuzzy inference system to precisely capture the entered preferences for ranking purpose.

2 Architecture

Our proposed architecture offers a unified solution that uniquely applies state of the art technologies of semantic services, agent negotiation, and multi-objective and constraints optimization to satisfy the requirements of whole service deployment life cycle. The main goal of the architecture is to provide: ease of use for non-experts, semantic interoperability, more precise discovery and selection, more reliable service level agreement (SLA) monitoring, and automatic negotiation strategy. The proposed architecture is depicted in Fig. 1 and its main components are described below:

1. *User portal.* All services provided by the system are presented via the web portal to clients. This component provides graphical interfaces to capture users' requirements such as software, hardware, quality of service (QoS) requirements (including maximum acceptable latency between tiers, minimum acceptable reliability, and budget), firewall, and scaling settings. In addition, it transforms user requirements to web service modeling ontology (WSMO) format in the form of goals which are then used for Cloud service discovery and composition. For more details regarding the format of goals, readers can refer to our previous work [3]. Moreover, it contains an account manager, which is responsible for user management.
2. *Translator.* Since web service modeling ontology is used for service discovery, Cloud services information is translated to web service modeling language format by the Translator component. This component takes care of building and maintaining an aggregated repository of Cloud services and is explained in detail in Section 2.1.
3. *Cloud service repositories.* They are represented by appliance and virtual unit service repositories in Fig. 1 and allow IaaS providers to advertise their services. For example, an advertisement of a computing instance can contain descriptions of its features, costs, and the validity time of the advertisement. From standardization perspective, a common metamodel that describes IaaS provider's services has to be created. However, due to the lack of standards, we developed our own metamodel [3] based on previous works.
4. *Discovery and negotiation service.* This component maps user's requirements to resources using the ontology-based discovery technique. It acts in user's interest to satisfy quality of service requirements by selecting the set of eligible IaaS providers. The negotiation service uses a time-dependent negotiation strategy that captures preferences of users on QoS criteria to maximize their utility functions while only accepting reliable offers. These negotiation strategies are described in our previous work [4].
5. *Composition optimizer.* Once the negotiation completed and eligible candidates are identified, the composition component, which is the focus of this paper, builds the possible compositions and uses the compatibility checking component to eliminate incompatible candidates. Then composition optimizer evaluates the composition candidates using the users' QoS preferences. The composition optimizer takes advantage of multi-objective algorithm and fuzzy logic to let users express their preference conveniently, and efficiently selects the closest candidates to users' interests. Throughout the paper we show how ontology-based compatibility checking, multi-objective algorithms, and fuzzy logic techniques can work in harmony to provide an elegant solution to the composition problem.
6. *Planning.* The planning component determines the order of appliance deployment on the selected IaaS providers and plans for the deployment in the quickest possible manner.
7. *Image packaging.* The Packaging component builds the discovered virtual appliances and the relevant meta-data into deployable packages, such as Amazon Machine Image (AMI) or Open Virtualization

Format (OVF) [5] packages. Then the packages are deployed to the selected IaaS provider using the deployment component.

8. *Deployment component*. It configures and sets up the virtual appliances and computing instances with the necessary configurations such as firewall and scaling settings. For example in a web application, specific connection details about the database server need to be configured.
9. *Deployment descriptor manager*. This component persists specifications of required services and their configuration information such as firewall and scaling settings in a format called deployment descriptor. Besides, it includes the mapping of user requirements to the instances and appliances provided by the Cloud. The mapping includes instance description (e.g., name, ID, IP, status), image information, etc. This meta-data is used by the appliance administration service to manage the whole stack of Cloud services even if they are deployed across multiple Clouds. Formally described using WSML, the deployment descriptor is located in our system (as a third party service coordinator), and in a Cloud-independent format that is used for discovering and configuring alternative deployments in case of failures. An example of a deployment descriptor is shown in Appendix B (supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCC.2014.2300855). It identifies how firewall and scaling configurations have to be set for Web server appliances. In addition, Deployment Descriptor helps to describe the utility function of users for provisioning extra Cloud services when scaling is required. This helps to create scaling policies that utilize the optimization component on the fly to provision services that maximizes the user's utility functions. For example, providers that have the lowest price, latency, and highest reliability are going to be ranked higher.
10. *Appliance administration service*. After the deployment phase, this component helps end users to manage their appliances (for example starting, stopping, or redeploying them). It uses the deployment descriptor to manage the deployed services.
11. *Monitoring and SLA management*. This component provides health monitoring of deployed services and provides required inputs and data for failure recovery and scaling. A monitoring system is provided by this component for fairly determining to which extent an SLA is achieved. More details on this component is provided in our previous paper [6].
12. *Failure recovery*. It automatically backs up virtual appliance data and redeploys them in the event of Cloud service failure.
13. *Decommissioning*. In the decommissioning phase, Cloud resources are cleaned up and released by this component.
14. *IaaS providers*. They are in both fabric and unified resource level [7] and contain resources that have been virtualized as virtual units. Virtual unit can be a virtual computer, database system, or even a virtual cluster. In addition to virtual units, they offer virtual appliances to satisfy software requirements of users.

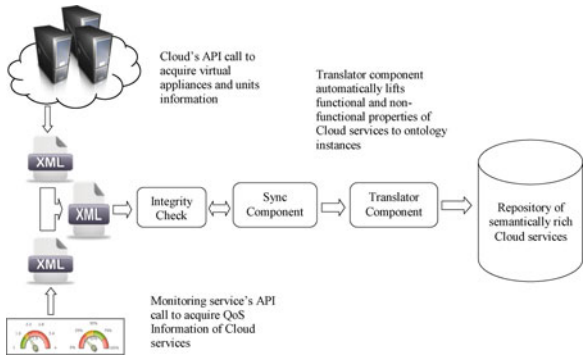


Fig. 2. The process of translation of the virtual appliances and units descriptions to WSML.

The first step in service composition is modeling the Cloud services (based on cost, size, functionality, etc.) and QoS requirements of users. This step not only allows appliance and virtual unit providers to advertise QoS of their services, but also provides a way for end users to express their QoS preferences. In this work, WSMO is extended and used for appliance and virtual unit QoS modeling. This allows us to model compatibility and legal constraints required to build a valid composition. However, the process of converting the Cloud service advertisements to WSML is time-consuming and error-prone if it is carried out manually. Automatic construction of WSMO-based service description from IaaS service advertisements is described in the next section.

2.1 Construction of Semantic Cloud Services

Currently, there is no integrated repository of semantic-based services for virtual appliances and units. The first step towards describing services and their QoS is to communicate with Clouds and the Cloud monitoring services through their APIs and gather required meta-data for building the repository. The process of metadata translation is demonstrated in Fig. 2. The components involved in this process are:

2.1.1 Integrity Checking

This component first merges output messages of API calls for acquiring Cloud services description using Extensible Stylesheet Language Transformations (XSLT)[1] and then compares them with the previously merged messages using a hash function. If the outputs of the hash function are not equal, the component triggers the Sync component to update the semantic repository.

2.1.2 Sync Component

The goal of this component is to keep the semantic-based repository consistent with the latest metadata provided by Cloud providers. As the synchronization is computing intensive, it is avoided unless the integrity checking component detects any inconsistency. It receives the output message that is required for synchronization and finds the

1. XSLT. http://www.w3.org/TR/xslt.

corresponding semantically rich services and updates them with the output of the translator component.

2.1.3 Translator Component

During the communication of a semantic-level client and a syntactic-level web service, two directions of data transformations (which is also called grounding) are necessary: the client semantic data must be written in an XML format that can be sent as a request to the service, and the response data coming back from the service must be interpreted semantically by the client. We use our customized Grounding technique on WSDL operations (that are utilized to acquire virtual appliance and unit metadata) output to semantically enrich them with ontology annotations. WSMO offers a package, which utilizes Semantic Annotations for WSDL (SAWSDL) for grounding [8]. It provides two extensions attribute namely as Lifting Schema Mapping and Lowering Schema Mapping. Lowering Schema Mapping is used to transfer ontology to XML and lifting Schema Mapping does the opposite. In our translator component, the lifting mapping extension has been adopted to define how XML instance data that is obtained from Clouds API calls is transformed to a semantic model.

As the first step in grounding, from output message schema, the necessary ontology is created for virtual units and appliances. The basic steps to build the ontology from XML schema using WSMO grounding is explained by Kopecký et al. [8]. In this step our contribution lies on building the ontology from multiple output message schemas. It means that the monitoring service output message schema is used to extend the ontology to encompass non-functional properties. This can be accomplished by merging two schemas to construct an output message that describes the format of the elements that has functional and non-functional properties such as price and reliability.

Having the ontology available, the next step is to add the necessary Mapping URI for all element declarations. For this purpose Modelreferences are used, which are attributes whose values are lists of URIs that point to corresponding concepts in the constructed ontology. Subsequently, we need to add schema mappings that point to the proper data lifting transformation between XML data and semantic data. For this purpose, two attributes, namely liftingSchemaMapping and loweringSchemaMapping, are offered by SAWSDL. These aforementioned attributes are then utilized to point from Cloud virtual appliance meta-data schema to a XSLT, which shows how meta-data is transferred from XML to WSML.

We tested this approach for Cloud service repositories with variety of sizes, and will present the experimental result in Section 5.2.2. The ontology listed in Appendix A (supplemental materials, available online) was partially created by the described translator component. For example, it shows how an appliance meta-data with ID of "ki00806369" has been translated to WSMO format.

Semantic service toolkits and libraries based on OWL-S and WSMO use XML based grounding. This XML mapping approach cannot deal with the growing number of Cloud provider's interfaces that use non-SOAP and non-XML services. The main reason that we have used XML is to follow the path that was suggested by WSMO, standard libraries and documentation provided by WSMO, and that major IaaS providers currently have a full support for XML based services. For alternative approaches of grounding for non-XML services readers can refer to studies conducted by Lambert et al. [9].

3 Evaluation of Composition Criteria

The composition problem is to find the best combination of compatible virtual appliances and virtual machines that minimizes the deployment cost (DC) and deployment time (DT), and maximizes the reliability while adhering to composability constraints. A formal description of the problem is given below.

3.1 Provider and User Request Model

Let m be the total number of providers. Each provider can provide virtual appliances, virtual units or both, and is represented as shown in the following equation:

$$\begin{aligned} \text{Provider } P_k : \{\{a\}, \{vm\}, C_{d_{ext}}, C_{d_{int}}\} \\ where\ \ 0 < k \leq m, \end{aligned} \tag{1}$$

where a, vm, $C_{d_{ext}}$, $C_{d_{int}}$ denotes appliance, virtual machine, Cost of external data transfer and Cost of internal data transfer, respectively. A virtual appliance a can be represented by a tuple of five elements (Equation (2)): appliance type, cost, license type, compatibility list, and size.

$$\begin{aligned} a : \{ApplianceType,\ Cost, LicenseType, \\ CompatibilityList,\ Size\}. \end{aligned} \tag{2}$$

A virtual machine vm can be formally described as a tuple with two elements as shown in the following equation:

$$vm : \{MachineType,\ Cost\}. \tag{3}$$

The user request for the appliance composition can be translated into a weighted graph $G(V, E)$ where each vertex represents a server that consists of a virtual appliance running on a virtual machine. Server corresponding to vertex v is represented by

$$S_v = \{a_v, vm_v\}, \quad \forall v \in V. \tag{4}$$

Each edge $e\{v, v'\} \in E$ indicates that corresponding servers communicate. The Data Transfer Rate between these connected vertexes (i.e., one server to another) is given by the weight associated to edge e.

3.2 Compatibility

When multiple Cloud services (i.e., virtual appliances and units) are composed together, they should be compatible with each other. In this work, we consider legal and image format compatibility constraints. However, it should be noted that in reality there will be other compatibility constraints such as compatibility between the products installed on the appliances.

- *Virtual appliance image format compatibility.* Before we finalize the deployment plan, we have to find out whether the image formats of chosen set of virtual appliances are compatible with the destination virtual unit provider. As it is illustrated in the first row of Table 1, a sample of ontology-based

TABLE 1
Compatibility Reasoning for Cloud Service Composition

Query String	Result
?x[isCompatipleWith hasValue ?y] :- ?x memberOf virtualAppliance and ?x [hasName hasValue ?v] and ?v=aki00806369 and ?y memberOf virtualUnit and ?y[hasProvider hasValue?pvu] and ?pvu [supportVmFormat hasValue ?supportedFormat] and ?x[hasFormat hasValue ?format] and ?format[hasName hasValue ?formatName] and ?supportedFormat[hasName hasValue ?supportedFormatName] and ?supportedFormatName=?formatName.	*?formatName: "AMI" ?v: "aki00806369" ?pvu: AmazonCalifornia ?p: AmazonCalifornia ?supportedFormatName: "AMI" ?format: AMI ?supportedFormat: AMI ?y: largeInstance ?x: aki00806369*
?x [isCompatipleWith hasValue ?y]:- ?x memberOf virtualAppliance and ?x [hasName hasValue ?v] and ?v=aki00806369 and ?x[hasProvider hasValue ?p] and ?y memberOf virtualUnit and ?y[hasProvider hasValue ?pvu] and ?p[hasCountry hasValue ?capp] and ?pvu [hasCountry hasValue ?cvu] and?capp[hasName hasValue ?cappName] and ?cvu[hasName hasValue ?cvuName] and ?cappName=?cvuName.	*?v: "aki00806369" ?pvu: AmazonCalifornia ?p: AmazonCalifornia ?capp: USA ?cappName: "USA" ?cvu: USA ?cvuName: "USA" ?y: largeInstance ?x: aki00806369*

reasoning on the built knowledgebase (as shown in Appendix A, available in the online supplemental material) shows that image with the ID of "aki00806369" is compatible with the large instance type provided by Amazon EC2.

- *Legal requirements.* In Cloud infrastructure, virtual machines can be deployed in data centers located in different parts of the world. However, there are legal requirements, for example US restrictions on exporting encryption technology [11], that prevent the export and deployment of software developed in one country to another. Hence, we need to ensure that the virtual appliances can be legally deployed on the selected virtual units. The second row of the Table 1 presents a query which sets appliance with the ID of "aki00806369" compilable to only virtual units provided by Clouds located in a same country where the appliance provider is situated.

To evaluate the compatibility requirements, first the ontology-based vocabulary is created using WSML, as shown in the Appendix A [line 225], available in the online supplemental material, where compatibility constraints are imposed by experts in the form of axioms in the ontology or alternatively by reasoning on the ontology. For example, the ontology listed in Appendix A, available in the online supplemental material, shows how an axiom (set by an expert) enforces that appliances can only be deployed on virtual units provided by Cloud providers that are located in the same country as the appliance provider. After building the required ontology, we can exploit the advantages of description logic (DL) to query the knowledgebase and check compatibility constraints of composition candidates. A WSML-DL query sample, given in Table 1, shows which virtual unit is compatible (legally and regarding image format) to the appliance with the ID of "aki00806369". Our objective is to achieve full compatibility among the appliances in the composition. Based on the compatibility constraints considered in our work, the compatibility (C) can be calculated based on the following equation:

$$C = \begin{cases} 0, & \text{if there exists at least one pair of} \\ & \text{incompatible services;} \\ 1, & \text{otherwise.} \end{cases} \quad (5)$$

Algorithm 1: Compatibility evaluation algorithm

```
Input: Composition c, Constraint List (cl) such as
       Image Format and Legal Compatibility
Output: Composition Validity
if CompositionValidity(c, cl) Exists in Cache then
    ValidComposition =
    GetCompositionValidityFromCache(c, cl);
end
ValidComposition=True;
foreach Appliance a and Virtual Unit vu In c do
    foreach Constraint t in cl do
        if Compatibility (t, a, vu) Exists in Cache
        then
            ValidComposition =
            GetCompatibilityFromCache(t, a, vu);
        end
        else
            ValidComposition =
            CheckCompatibilitybyReasoning(t, a,
            vu);
        end
        InsertCompatibilitytoCache(t, a, vu);
    end
    if ValidComposition=False then
        break;
    end
end
InsertCompositionValiditytoCache(c, cl);
return ValidComposition;
```

In addition, Algorithm 1 illustrates the process to evaluate the compatibility of the services in a composition. The

algorithm checks the compatibility constraint (by sending the related query to WSML-reasoner) for each pair of appliance and virtual unit in the composition, and only compositions where all constraint queries are satisfied would be returned as valid. We use WSMO discovery component for service discovery and implemented the Algorithm 1 to discard sets of incompatible discovered services.

3.3 Cost

The costs involved in procuring and using virtual appliances can be categorized as follows:

- *Acquisition cost.* Costs involved in purchasing the virtual appliance, such as licensing cost, cost of the virtual machine and any costs associated with deployment such as the data transfer costs to transfer the appliances to the virtual machine at the IaaS provider.

  To build a server S_v, let us assume appliance a_v is obtained from provider P_k and virtual machine vm_v is obtained from provider P_l. If this server runs for a duration of T (lease period), the Equation (6) shows the acquisition cost

$$AqCost(S_v) = Cost(a_{v,P_k}) \times T + Cost(vm_{v,P_l}) \times T + AppTransCost(k,l), \tag{6}$$

  where $Cost(a_{v,P_k})$ is the cost of appliance per unit of time, $Cost(vm_{v,P_l})$ is the cost of virtual machine per unit of time, and the cost of appliance transfer from appliance provider k to virtual unit provider l is given by

$$AppTransCost(k,l) = \begin{cases} 0, & \text{if } k = l; \\ Size(a_{v,P_k}) \times C_{d_{ext}}(P_l), & \text{if } k \neq l. \end{cases} \tag{7}$$

- *Ongoing cost.* This will include the costs of running the virtual appliance, such as the cost of data transfers. In this work we consider only the costs associated with data transfers as ongoing costs.

  Let v, v' be two vertexes (servers) on provider l, connected by edge $e\{v,v'\}$. The data transfer cost between the two servers is given by

$$TransCost(e\{v,v'\}) = Size(Data_{e\{v,v'\}}) \times C_{d_{int}}. \tag{8}$$

- *Decommissioning cost.* Decommissioning cost primarily includes archival and removal costs of the data at the end of the application life cycle such as the data sanitisation, and henceforth as shown in Equation (9) will depend on the size of the data stored. The amount of data stored will vary from server to server

$$DecomCost(S_v) = CostPerUnit \times SizeOfData_v. \tag{9}$$

Based on the costs calculated above, total cost (TC) can be computed as shown in the following equation:

$$TC = \sum_{v \in V} AqCost(S_v) + \sum_{e \in Ev, v' \in V} TransCost(e\{v,v'\}) + \sum_{v \in V} DecomCost(S_v). \tag{10}$$

3.4 Deployment Time

Virtual appliances significantly minimize the time required to build and configure the necessary independent components. However, size of virtual appliances ranges from a few megabytes to tens of gigabytes depending on the applications installed on them and will impact the time required to transfer and deploy the appliances from the appliance provider to the virtual machine provider. Hence, we consider the deployment time as one of the composition objectives. The time required to deploy a given appliance a_v obtained from provider P_k on a virtual machine vm_v obtained from provider P_l is given by

$$DT(a_{v,P_k}, vm_{v,P_l}) = \begin{cases} InitTime(vm_{v,P_l}), & \text{if } k = l; \\ InitTime(vm_{v,P_l}) + \frac{Size(a_{v,P_k})}{DataTransferRate(P_k,P_l)}, & \text{if } k \neq l, \end{cases} \tag{11}$$

where $InitTime(vm_{v,P_l})$ is the time required to initialize the appliance on the virtual unit. As appliances can be deployed in parallel, total deployment time will be given by

$$TD = Max\{DT(a_{v,P_k}, vm_{v,P_l})\}. \tag{12}$$

3.5 Reliability

For measuring Cloud providers reliability, we introduce SLA Confidence Level (SCL), which is a metric to measure how reliable services of each provider are based on the SLAs and their performance history. SCL values are computed by a third party that is responsible for monitoring the SLA of providers based on the following equation:

$$SCL = \sum_{j=1}^{q} (I_j \times SCL_j), \tag{13}$$

where the SCL_j is SLA confidence level for QoS criteria j of a Cloud service; I_j is the importance of the criteria j for the user; q is the number of monitored QoS criteria.

We utilized the beta reputation system [12] to assess the SCL for each criterion. The reason is that the Monitoring Outcome (MO_{jt}) of a particular quality of service criteria j in the period of t in the SLA contract can be modeled as shown in Equation (14), and therefore it is a binary event. Consequently, the beta density function, which is shown in Equation (15), can be efficiently used to calculate posteriori probabilities of the event. As a result, the mean or expected value of the distribution can be represented by Equation (16):

TABLE 2
Virtual Appliance Composition Objectives

Criteria	Metric	Type	Requirement
Compatibility (C)	C	Constraint	To be equal to 1
Total Cost (TC)	$	Objective	To be minimized
Total Deployment Time (TD)	mS	Objective	To be minimized
Total Reliability (TR)	SCL	Objective	To be maximized

$$MO_{jt} = \{SLAnotviolated,\ SLAviolated\}, \tag{14}$$

$$f(x|\rho,\tau) = \frac{\Gamma(\rho+\tau)}{\Gamma(\rho)\Gamma(\tau)} x^{\rho-1}(1-x)^{\tau-1} \quad where\ 0 \le x \ge 1, \rho < 0, \tau > 0,\ and\ \rho\ and\ \tau\ are\ beta\ distribution\ parameters. \tag{15}$$

$$\mu = E(x) = \rho/(\rho+\tau). \tag{16}$$

As mentioned earlier in Section 2, in our architecture a component is responsible for monitoring SLA contracts. If we assume that the monitoring component has detected that SLA violation occurred v times for provider of p (for total number of n monitored SLAs). Consequently, considering that $p = n - v + 1$, $\tau = v + 1$ and, the SCL is equal to the probability expectation that the SLA is not going to be violated and is calculated as shown in the following equation:

$$SCL_j = \frac{n - v + 1}{n + 2}. \tag{17}$$

We modeled availability for SCL generation, as current Cloud providers only include "availability": in their SLAs. The reliability in our work is considered as a user constraint for each Cloud service.

3.6 Overall Objectives

The final objective is to find a fully compatible service composition that minimizes the deployment cost and time, and improves the reliability while adhering to compatibility constraints as shown in Table 2.

4 Composition Optimization

In our problem we consider three user objectives, the lowest cost, quickest deployment time, and the highest reliability. This makes it infeasible to find an optimal composition as these objectives can conflict with each other. One way to address this problem is to convert the appliance composition problem to a single-objective problem by asking users to give weights for all the objectives. However, this approach is error-prone and impractical, as not all the users have the knowledge to accurately assign weights to objectives. Furthermore, since the composition solutions will depend on the capability of users to assign proper weight to the objectives, additionally we have to find a way to evaluate the knowledge of users about each objective to ensure the accuracy of the approach. As shown in Fig. 3, we have tackled these challenges in two steps:

- First, we find the Pareto front [13] composition solutions using different multi-objective algorithms (OMOPSO, NSGA-II, and SPEA-II). We have used Jmetal [14] for this purpose which allows us to define problems by defining each gene (variable) to point to Cloud service candidates as shown in Fig. 3. Then, we defined necessary fitness functions for all the defined objectives and choose the algorithm to solve the problem.
- Second, we help users to describe their preferences using high level "if-then" rules, which builds our fuzzy engine rule-base to rank the Pareto front acquired from the previous step. By doing this we acquire Pareto front once and then filter it according to the user preferences. This has a great advantage when compared to the method offered in [15] as our method does not require to search the solution space each time the user preferences change.

Evolutionary algorithms have been effectively applied for solving optimization problems. Among them NSGA-II [16] and SPEA-II [17] outperform many other genetic optimization algorithms [16]. Nevertheless, recently other

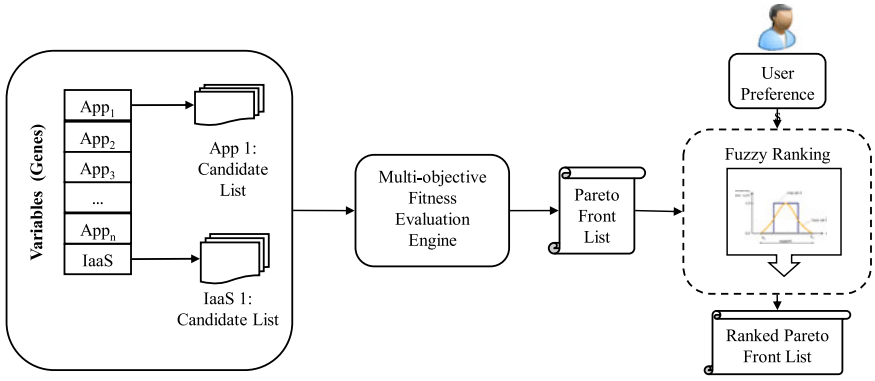
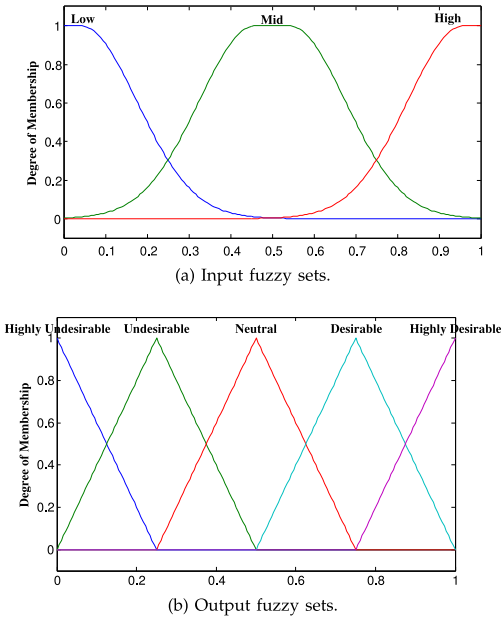


Fig. 3. Cloud service composition optimization approach.



(a) Input fuzzy sets.

(b) Output fuzzy sets.

Fig. 4. Fuzzy engine input and output fuzzy sets.

meta-heuristics that work based on swarm intelligence such as Particle Swarm Optimization (OMOPSO) [18] are also used to tackle multi-objective optimization problems. In this work, we perform a comparative study among the algorithms NSGA-II, SPEA-II, and OMOPSO for the appliance composition problem to determine which of them will be suitable for our problem.

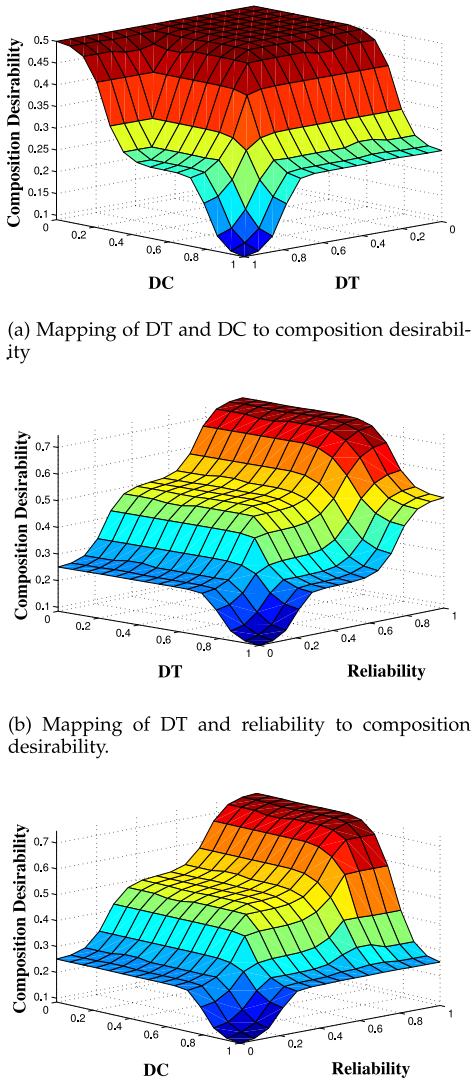
Fuzzy logic based ranking. Our proposed fuzzy inference engine includes three inputs and one output. Inputs of the system are normalized deployment time, deployment cost, and reliability of composition, which are all described based on the same membership functions in Fig. 4a. Output of the fuzzy engine as shown in Fig. 4b represents how desirable the current set of inputs are based on the fuzzy rule-based indication. It shows the membership function for output by which we allow the gradual assessment of the membership of elements in a set. For example, the value "0" in output means the solution is highly undesirable whereas the value "1" shows that the solution is highly desirable. Fuzzy rules should be defined by the user to describe their preferences. For example a rule can be defined as:

if **DT** is *low* and **DC** is *low* and **Reliability** is *high*, composition is **highly desirable**.

Table 3 shows sample rules that can be expressed by users. In this work we use a fuzzy engine based on Mamdani inference system [19] with Centroid of area defuzzification strategy. Readers can refer to [20] for detailed information on fuzzy inference systems. Once rules and defuzzification strategy are defined, a fuzzy inference system can map the inputs of fuzzy engine to its output. Fig. 5 demonstrates how deployment time, cost, and reliability are mapped to composition desirability when all rules are set.

TABLE 3
Sample High Level Rules Set by Users

DC	DT	Reliability	Composition Desirability
Low	Low	Low	Undesirable
Low	Low	High	Highly Desirable
High	Low	High	Not sure



(a) Mapping of DT and DC to composition desirability

(b) Mapping of DT and reliability to composition desirability.

(c) Mapping of DC and reliability to composition desirability.

Fig. 5. Mapping from QoS criteria values to composition desirability value.

5 Performance Evaluation

In order to realize and evaluate the proposed approach, a number of components and technologies are utilized. Most importantly, multi-objective algorithms are implemented in jMetal [14]. After that, Pareto Front composition solutions from Jmetal have been passed to our fuzzy-logic based ranking components, which utilizes jFuzzyLogic [21] to define the membership functions and rules.

5.1 Case Study and Data Collection

We consider a case study of a web-based collaboration application for evaluating performance. The application allows users to store, manage, and share documents and

drawings related to large construction projects. The service composition required for this application includes: Firewall (x1), Intrusion Detection (x1), Load Balancer (x1), Web Server (x4), Application Server (x3), Database Server (x1), Database Reporting Server (x1), Email Server (x1), and Server Health Monitoring (x1). To meet these requirements, our objective is to find the best Cloud service composition.

5.1.1 Cloud Service Provider Information

Cloud service providers details, such as the promised availability and monitored availability (for calculating SCL), data transfer costs, and data throughput between two Cloud service providers (to estimate the transfer time of virtual appliances) are obtained using the CloudHarmony service.[2] When the required data is not available from them, it is directly obtained from the Cloud service provider. Similar to the virtual unit and appliance directory, the module to obtain the Cloud service provider information also can be used to update and extend the information in future.

5.2 Results

There are three main experiment results presented in this section: 1) an investigation of performance of the translator component, 2) a performance comparison between the OMOPSO, NSGA-II, and SPEA-II algorithms for the real case study, and 3) examination of the effectiveness of fuzzy inference system for handling imprecise user preferences. NSGA-II and SPEA-II algorithms use a population of size of 100, and maximum number evaluations of 25,000. OMOPSO is configured with 100 particles, with a maximum of 100 leaders and maximum number of iterations of 250. We have carried out 40 independent runs per experiment and then statistically analyzed results.

5.2.1 Performance of the Translation Approach

Major Cloud providers have large repository of virtual appliance and unit services. For example, size of Amazon Web Service repository[3] alone is greater than 10.6 megabytes. To increase the efficiency of the translation approach we only synchronize when the translation service is triggered by integrity checking component. We increased the number of services in the repository by merging repositories from various Cloud providers to investigate the scalability of our approach in terms of execution time needed for the translation. For each case of repository size, we repeated the experiment 30 times and the results are plotted in Fig. 6. Regression analysis shows that there is positive and linear relationship between the repository size and the translation time. The evidence confirms that the regression coefficient is 0.6621, which suggests that if the data size to be translated increases by a megabyte, translation time increases roughly by 0.6 second. Consequently, the synchronization function can be executed online in an acceptable time even if a considerable percentage of virtual appliance and unit properties are updated.

2. CloudHarmony. http://cloudharmony.com/.

3. The virtual appliance repository can be obtained by calling DescribeImages service from S3.amazonaws.com/ec2-downloads/ec2.wsdl.

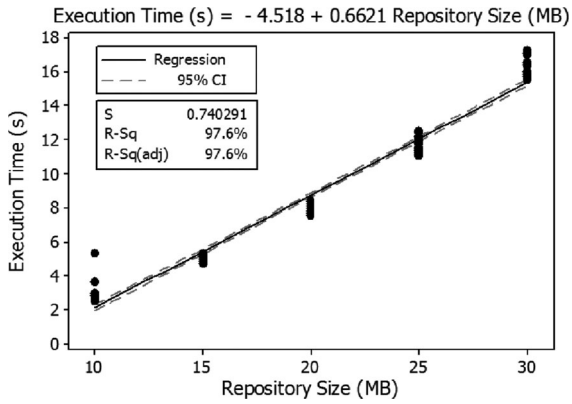


Fig. 6. Execution time of translation for different repository sizes.

5.2.2 Performance of the Optimization Approach

Performance of single-objective optimization algorithms can be evaluated by analyzing the best value achieved by the algorithms. However, in the case of multi-objective optimization, it is not practical. There are quality indicators, which can be used to evaluate the quality of the obtained set of solutions, and determine the convergence and diversity properties of algorithms. In our experiments, the algorithms are compared using execution time; inverse generational distance (IGD) [17], which determines convergence of algorithms; Spread, which determines diversity [22]; and Hypervolume, which determines both. In addition, it has been complemented by the application of statistical tests to ensure the significance of the results. Otherwise, the drawn conclusions may be incorrect as the differences between the algorithms could have occurred by chance. As in our problem the Pareto fronts are not known, applying the aforementioned quality indicators is not possible. To tackle this issue and as a common approach, we build a reference front by collecting all the results of 100 runs of the algorithms. This helps us to compare the relative performance of algorithms (Jmetal [14] provides an automatic way of obtain the reference front).

We start with describing the Hypervolume [23] indicator which has been widely used and calculates the volume of dominated portion of the objective space. The indicator is strictly Pareto-compliant [24] which means if it values a solution higher than the other, then that solution set dominates the other one's. In addition to Hypervolume, we have also used other two widely used and recommended indicators [22], namely IGD and spread. As shown in Equation (18), IGD works out the average distance of the obtained solution points from the optimal Pareto fronts (reference front in our case), where the d_i is the Euclidean distance between the obtained solution points and the closest member of optimal Pareto front and n is the number of points in the optimal Pareto front. Hence, when the achieved solution is in the optimal Pareto front the IGD is equal to 0:

$$IGD = \frac{\sqrt{\sum_{i=1}^{n} {d_i}^2}}{n}. \tag{18}$$

In addition, spread is a metric to calculate the broadness and calculated based on the following equation:

TABLE 4
Statistical Comparison of Algorithms

95% CI for Difference	ANOVA Test			
	Execution time	**Spread**	**IGD**	**Hypervolume**
NSGA-II-SPEA-II	(-2469.5,-2579.4)	(0.0476,0.1820)	Not Significant	Not Significant
NSGA-II-OMOPSO	(420.4, 531.1)	(0.2002,0.3355)	(0.003222,0.009074)	(0.00599,0.01848)
SPEA-II-OMOPSO	(2944.8, 3055.5)	(0.0854,0.2206)	(0.002913,0.008765)	(0.00372,0.01620)

$$Spread = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \overline{d}|}{d_f + d_l + (N-1)\overline{d}}, \quad (19)$$

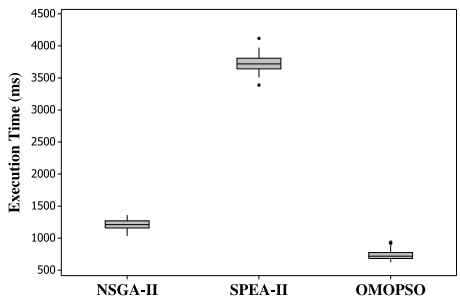
where d_i is the Euclidean distance between consecutive solutions in the obtained solutions. d_f and d_l are the Euclidean distance between the boundary solutions of the obtained pareto front set. When spread is equal to zero, it means that the obtained solutions are well diverse.

The spread, IGD, Hypervolume, and execution time of three algorithms are compared using analysis of variance (ANOVA) test, as the quality values are normally distributed and there is no strong evidence to indicate that variance is not constant. From the ANOVA table the P-value is less than 0.001, which strongly suggests that there are differences in mean spread, IGD, Hypervolume, and execution time between the algorithms. The algorithm that obtains smaller spread is capable of acquiring a set of non-dominated composition solutions with better diversity. In addition, if an algorithm achieves smaller value for IGD, it is better in converging to Pareto-optimal front. However, algorithms with larger value of Hypervolume are more desirable.

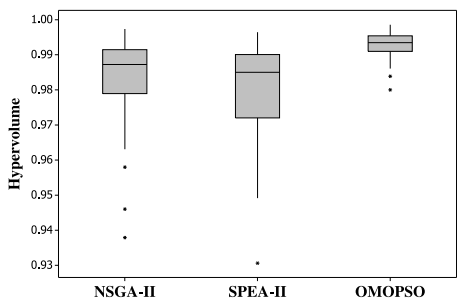
In our experiment, when Hypervolume is used for comparison, OMOPSO outperforms NSGA-II and SPEA-II as shown in Table 4 and Fig. 7b. In addition, if spread is used for diversity comparison, as shown in Table 4 and Fig. 7c, 95 percent confidence interval (CI) for the difference between NSGA-II - OMOPSO and SPEA-II - OMOPSO are (0.2002, 0.3355) and (0.0854, 0.2206) respectively. This shows the better suitability of OMOPSO in achieving higher diversity. In addition, SPEA-II outperforms NSGA-II in terms of Spread. Moreover, as Fig. 7a illustrates OMOPSO has the lowest execution time. Furthermore NSGA-II performs better than the SPEA-II in terms of execution time. For the convergence comparison, IGD has been considered. As shown in Fig. 7d and in Table 4, 95 percent CI for differences between NSGA-II - OMOPSO and SPEA-II - OMOPSO are reported as (0.003222, 0.009074) and (0.002913, 0.008765) respectively, which shows OMOPSO is better in converging to Pareto-optimal front.

Fig. 8a shows the output of NSGA-II for the case study appliance composition without considering user preferences. In this case users would receive a set of non-dominated composition solutions. It indicates that all the composition solutions have their own outstanding characteristics and none of the points could dominate the others.

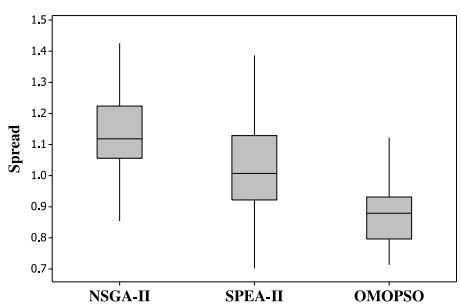
In reality, Cloud users at least have some vague idea regarding their objectives which can be captured by asking them to set high level linguistic rules. To emulate the user behavior, in our experiment, 27 sample fuzzy rules are designed similar to the one described in Section 4. Based on these rules, by applying fuzzy inference system, we mapped points in Fig. 8a to a number between 0 (least desirable) and 1 (most desirable). That number was used to color the points



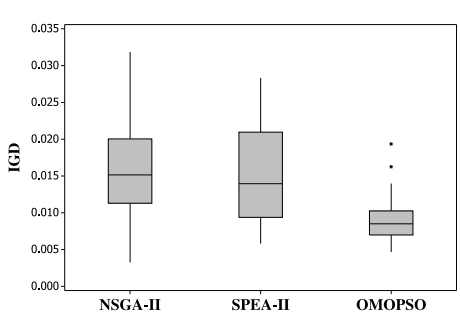
(a) Execution time.



(b) Hypervolume.

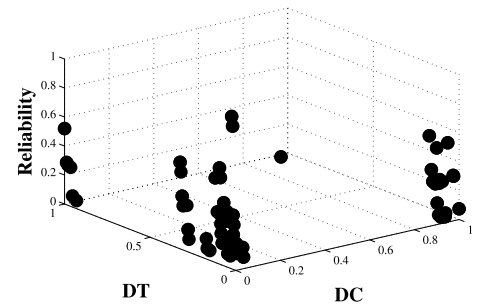


(c) Spread.

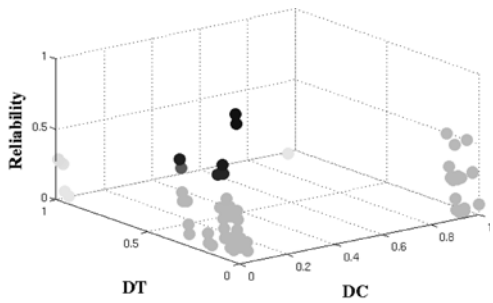


(d) IGD.

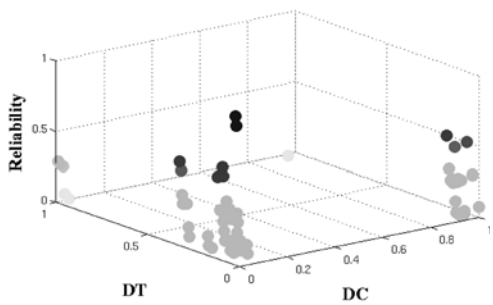
Fig. 7. Comparison of algorithms using quality indicators.



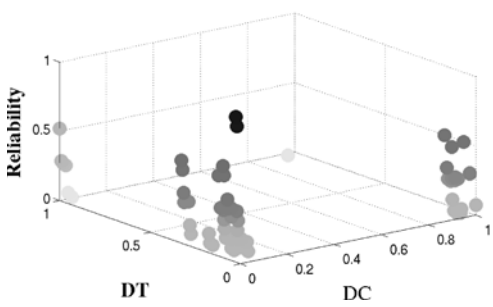
(a) NSGA-II pareto front appliance composition solutions without any user preferences.



(b) Coloring NSGA-II output using fuzzy Logic based on 9 user preference rules.



(c) Coloring NSGA-II output using fuzzy Logic based on 18 user preference rules.



(d) Coloring NSGA-II output using fuzzy Logic based on 27 user preference rules.

Fig. 8. Appliance composition optimization results.

in a way that if the composition solution is more appealing to the user, the corresponding point is darker. As shown in Figs. 8b and 8c when the number of preferences defined by the user using the if-then rule increases, the solution's prominence also becomes clearer. As it can be seen in Fig. 8d, when all the 27 rules are set (none of them set as "not sure"), the majority of points have distinct colors, however as the number of rules set to "not sure" increases more and more solutions will have the same color. As Fig. 8d shows two points at the center are the darkest and therefore are the most appealing solutions. Using these techniques non-expert Cloud users are now able set their preferences using high level if-then rules and get user friendly recommendations on the solution's prominence. This experiment also shows how the system could be helpful for end user with different levels of knowledge of the preferences. It is worth mentioning that traditionally when end users are asked to assign weights to the objectives, they are supposed to have full understanding of their preferences.

6 Related Work

Konstantinou et al. [25] proposed an approach to plan, model, and deploy Cloud service compositions. In their approach, the solution model and the deployment plan for the composition in Cloud platform are developed by skilled users and executed by unskilled users. Likewise, in our system, a set of compatibility constraints from experts were captured which would be utilized to simplify the process of deployment for end users by eliminating invalid compositions solutions. However, as they also mentioned, their work lacks an approach for appliance selection and their placement on the Cloud which is offered by our work. Similarly Chieu et al. [26] proposed the use of composite appliances to automate the deployment of integrated solutions. However in their work, QoS objectives are not considered.

Similarly another work has utilized intuitionistic fuzzy set (IFS) for ranking service compositions in the context of Grid and SOA [27]. It does not deal with users' constraints such as compatibility and when the problem is NP-hard (like our service composition problem) the execution time is not acceptable. Furthermore, in comparison with the work which considered evolutionary approach such as NSGA-II for service composition [13] our approach improves the composition solution diversity and convergence and decreases the execution time.

Unified Cloud Interface (UCI) provides ontology[4] model for modeling Amazon EC2 services. Mosaic project [28] is proposed to develop multi-Cloud oriented applications. In Mosaic, Cloud ontology plays and essential role, and expresses the application's needs for Cloud resources in terms of SLAs and QoS requirements. It is utilized to offer a common access to Cloud services in Cloud federations. However, none of these ontologies focus on modeling of compatibility of Cloud services.

There are several existing approaches [29], [30], [31], [32] that are capable of dealing with incompatible services. However, many of them only focused on compatibility of Input and Output (I/O) of services and did not consider incompatibilities that are caused by regulations and other factors that are not related to service functionalities.

In addition, OPTIMIS [33] main contribution is optimizing the whole service life cycle, from service construction,

4. OWL Ontology provided by OCI https://code.google.com/p/unifiedcloud/source/browse/trunk/ontologies/.

deployment, to operation in Cloud environments. The considered QoS criteria in OPTIMIS are trust, risk, eco-efficiency and cost. The evaluation of Cloud provider is accomplished through an adoption of analytical hierarchy process (AHP). In comparison with our approach for appliance composition, works that applied analytical hierarchy process and multi-attribute utility theory (MAUT) [34], can only perform well when the number of given alternatives is small and the number of objectives is limited. In contrast, our approach can deal efficiently with a large number of Cloud services in the repository.

While there are other studies [35], [36], [37], [38] that focus on appliance selection and deployment problem, we are not aware of any work that provides a framework for composing and deploying multiple virtual appliances with the focus on automatic compatibility checking and QoS-aware ranking.

7 Conclusions and Future Directions

In order to tackle Cloud service composition challenges, we presented an ontology-based approach to describe services and their QoS properties, which helped us to build a composition with a set of compatible services. Our system helps non-expert users with limited or no knowledge on legal and image format compatibility issues to deploy their services faultlessly. In addition, we proposed a technique to optimize the service composition based on user preferences such as deployment time, cost, and reliability. The approach exploits the benefits of evolutionary algorithms such as OMOPSO, NSGA-II, and SPEA-II for optimization and fuzzy logic to handle vague preferences of users. Results show that for the proposed case study, we can effectively help an unskilled user to identify the appliance compositions which are closest to their preferences.

Integer linear programming (ILP) [39], pseudo-Boolean optimisation (PBO) [40], and multiple objective ant colony optimization (MOACO) [41] are all competitive algorithms, and with the best of our knowledge, for this particular problem, there is no in-depth comparison between all three and OMOPSO. Therefore, providing such a comparison can be considered as another possible future work.

Moreover, Cloud services have specific characteristics and QoS dimensions which have to be identified. Specifically, defining criteria which are able to model energy and carbon emission efficiency, reliability, and trust of a Cloud service are increasingly attractive to users. For example, methods to evaluate reliability and trust of providers from user feedbacks and monitoring services together can be further studied. This consists of collecting required raw data from trusted sources and statistically analyzing and aggregating them. In addition, user experience is another important benchmark for Cloud service providers. Recently, crowdsourcing is being used to create collective knowledge to assess QoS. It requires an investigation on discovering the crowd that can evaluate a service efficiently, delegating evaluation tasks to crowds, and calculating the accuracy of the assessments.

The current implementation of the translator component supports XML-based Cloud services. Emerging specifications such as Open Cloud Computing Interface (OCCI) [10] aims at providing a standard way for describing Cloud resources. Therefore, investigating approaches that can semantically enrich these new formalisms can be taken as a future direction to further enhance the translator component.

Acknowledgments

We thank Yoganathan Sivaram, Adel Nadjaran Toosi and Rodrigo N. Calheiros for their constructive suggestions on this article.

References

[1] C. Sun, L. He, Q. Wang, and R. Willenborg, "Simplifying Service Deployment with Virtual Appliances," *Proc. IEEE Int'l Conf. Services Computing (SCC)*, 2008.

[2] J. De Bruijn, H. Lausen, A. Polleres, and D. Fensel, "The Web Service Modeling Language WSML: An Overview," *Proc. Third European Conf. Semantic Web: Research and Applications*, pp. 590-604, 2006.

[3] A. Dastjerdi, S. Tabatabaei, and R. Buyya, "An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery," *Proc. 10th IEEE/ACM Int'l Conf. Cluster, Cloud and Grid Computing*, 2010.

[4] A. Dastjerdi and R. Buyya, "An Autonomous Reliability-Aware Negotiation Strategy for Cloud Computing Environments," *Proc. 12th IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing*, 2012.

[5] DMTF, "Open Virtualization Format," http://www.dmtf.org/standards/ovf, 2014.

[6] A. Dastjerdi, S. Tabatabaei, and R. Buyya, "A Dependency-Aware Ontology-Based Approach for Deploying Service Level Agreement Monitoring Services in Cloud," *Software: Practice and Experience*, vol. 42, no. 4, pp. 501-518, 2011.

[7] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," *Proc. Grid Computing Environments Workshop (GCE)*, 2008.

[8] J. Kopeckỳ, D. Roman, T. Vitvar, M. Moran, and A. Mocan, WSMO Grounding. WSMO Working Draft v0. 1," 2007.

[9] D. Lambert, N. Benn, and J. Domingue, "Integrating Heterogeneous Web Service Styles with Flexible Semantic Web Services Groundings," *Proc. First Int'l Future Enterprise Systems Workshop*, 2010.

[10] T. Metsch, A. Edmonds, R. Nyrén, and A. Papaspyrou, "Open Cloud Computing Interface—Core," *Open Grid Forum*, http://forge. gridforum. org/sf/go/doc16161, 2010.

[11] R. Barth and C. Smith, "International Regulation of Encryption: Technology Will Drive Policy," *Borders in Cyberspace: Information Policy and Global Information Infrastructure*, pp. 283-299, MIT Press, 1999.

[12] A. Jsang and R. Ismail, "The Beta Reputation System," *Proc. 15th Bled Electronic Commerce Conf.*, 2002.

[13] Y. Yao and H. Chen, "QoS-Aware Service Composition Using NSGA-II1," *Proc. Second Int'l Conf. Interaction Sciences: Information Technology, Culture and Human (ICIS'09)*, pp. 358-363, 2009.

[14] J. Durillo and A. Nebro, "jMetal: A Java Framework for Multi-Objective Optimization," *Advances in Eng. Software*, vol. 42, no. 10, pp. 760-771, 2011.

[15] J. Branke and K. Deb, "Integrating User Preferences into Evolutionary Multi-Objective Optimization," *Knowledge Incorporation in Evolutionary Computation*, pp. 461-477, Springer, 2005.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, Apr. 2002.

[17] E. Zitzler and L. Thiele, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach," *IEEE Trans. Evolutionary Computation*, vol. 3, no. 4, pp. 257-271, Nov. 1999.

[18] M. Sierra and C. Coello, "Improving PSO-Based Multi-Objective Optimization Using Crowding, Mutation and E-Dominance," *Proc. Third Int'l Conf. Evolutionary Multi-Criterion Optimization (EMO)*, 2005.

[19] E. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int'l J. Man-Machine Studies*, vol. 7, no. 1, pp. 1-13, 1975.

[20] L. Zadeh, "Fuzzy Logic = Computing with Words," *IEEE Trans. Fuzzy Systems*, vol. 4, no. 2, pp. 103-111, May 1996.
[21] P. Cingolani and J. Alcala-Fdez, "jFuzzyLogic: A Robust and Flexible Fuzzy-Logic Inference System Language Implementation," *Proc. Int'l Conf. Fuzzy Systems*, pp. 1-8, 2012.
[22] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.
[23] D.A. Van Veldhuizen, "Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations," technical report, DTIC, 1999.
[24] C.A.C. Coello, G.B. Lamont, and D.A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, vol. 5, Springer, 2007.
[25] A.V. Konstantinou, T. Eilam, M. Kalantar, A.A. Totok, W. Arnold, and E. Snible, "An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds," *Proc. Third Int'l Workshop Virtualization Technologies in Distributed Computing*, pp. 9-18, 2009.
[26] T. Chieu, A. Mohindra, A. Karve, and A. Segal, "Solution-Based Deployment of Complex Application Services on a Cloud," *Proc. IEEE Int'l Conf. Service Operations and Logistics and Informatics*, 2010.
[27] P. Wang, "QoS-Aware Web Services Selection with Intuitionistic Fuzzy Set under Consumer's Vague Perception," *Expert Systems with Applications*, vol. 36, no. 3, pp. 4460-4466, 2009.
[28] B.Di Martino, D. Petcu, R. Cossu, P. Goncalves, T. Máhr, and M. Loichate, "Building a mosaic of clouds," *Proc. Euro-Par 2010 Parallel Processing Workshops*, pp. 571-578, 2011.
[29] P. Bartalos and M. Bieliková, "Automatic Dynamic Web Service Composition: A Survey and Problem Formalization," *Computing and Informatics*, vol. 30, no. 4, pp. 793-827, 2012.
[30] F. Rosenberg, M. Muller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar, "Metaheuristic Optimization of Large-Scale QoS-Aware Service Compositions," *Proc. IEEE Int'l Conf. Services Computing*, 2010.
[31] F. Lecue and N. Mehandjiev, "Towards Scalability of Quality Driven Semantic Web Service Composition," *Proc. IEEE Int'l Conf. Web Services*, pp. 469-476, 2009.
[32] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl, "A Scalable Approach for QoS-Based Web Service Selection," *Service-Oriented Computing—ICSOC 2008 Workshops*, Springer, 2009.
[33] M. Kiran, M. Jiang, D. Armstrong, and K. Djemame, "Towards a Service Lifecycle Based Methodology for Risk Assessment in Cloud Computing," *Proc. Ninth Int'l Conf. Dependable, Autonomic and Secure Computing*, 2011.
[34] V. Tran, H. Tsuji, and R. Masuda, "A new QoS Ontology and Its QoS-Based Ranking Algorithm for Web Services," *Simulation Modelling Practice and Theory*, vol. 17, no. 8, pp. 1378-1398, 2009.
[35] A. Aboulnaga, K. Salem, A. Soror, U. Minhas, P. Kokosielis, and S. Kamath, "Deploying Database Appliances in the Cloud," *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 13-20, 2009.
[36] R. Bradshaw, N. Desai, T. Freeman, and K. Keahey, "A Scalable Approach to Deploying and Managing Appliances," *Proc. TeraGrid Conference*, 2007.
[37] A. Rodriguez, J. Carretero, B. Bergua, and F. Garcia, "Resource Selection for Fast Large-Scale Virtual Appliances Propagation," *Proc. IEEE Symp. Computers and Comm.*, 2009.
[38] T. Pham, H. Truong, and S. Dustdar, "Elastic High Performance Applications—A Composition Framework," *Proc. IEEE Asia-Pacific Services Computing Conf.*, 2011.
[39] L. Zeng, B. Benatallah, A.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.*, vol. 30, no. 5, pp. 311-327, May 2004.
[40] V. Manquinho, J. Marques-Silva, and J. Planes, "Algorithms for Weighted Boolean Optimization," *Proc. 12th Int'l Conf. Theory and Applications of Satisfiability Testing*, pp. 495-508, 2009.
[41] D. Angus and C. Woodward, "Multiple Objective Ant Colony Optimisation," *Swarm Intelligence*, vol. 3, no. 1, pp. 69-85, 2009.

Amir Vahid Dastjerdi is a research fellow with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne. His current research interests include cloud service coordination, scheduling, and resource provisioning using optimization, machine learning, and artificial intelligence techniques. He is a member of the IEEE.

Rajkumar Buyya is future fellow of the Australian Research Council and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in cloud computing. He also serves as a visiting professor for the University of Hyderabad, India; King Abdulaziz University, Saudi Arabia; and Tsinghua University, China. He has authored more than 450 publications and four text books including "Mastering Cloud Computing" published by McGraw Hill and Morgan Kaufmann, 2013, for Indian and international markets, respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 78 and 28,000+ citations). He is a senior member of the IEEE.