QoS-Aware Cloud Service Composition Based on Economic Models

Zhen Ye[1,2], Athman Bouguettaya[3], and Xiaofang Zhou[1]

[1] The University of Queensland, Australia
[2] CSIRO ICT Centre, Australia
[3] Royal Melbourne Institute of Technology, Australia

Abstract. Cloud service composition is usually long term based and economically driven. We consider cloud service composition from a user-based perspective. Specifically, the contributions are shown in three aspects. We propose to use discrete Bayesian Network to represent the economic model of end users. The cloud service composition problem is modeled as an Influence Diagram problem. A novel influence-diagram-based cloud service composition approach is proposed. Analytical and simulational results are presented to show the performance of the proposed composition approach.

1 Introduction

Cloud computing is increasingly becoming the technology of choice as the next-generation platform for conducting business [1]. Big companies such as Amazon, Microsoft, Google and IBM are already offering cloud computing solutions in the market. A fast increasing number of organizations are already outsourcing their business tasks to the cloud, instead of deploying their own local infrastructures [2]. A significant advantage of cloud computing is its economic benefits for both users and service providers.

Cloud computing has been intertwined with SOC since its inception [3]. Service oriented computing (SOC) has been widely accepted as the main technology enabler for delivering cloud solutions [4]. Service composition is an active research area in service-oriented computing [5]. Compared to traditional service composition, cloud service composition is usually long-term based and economically driven. Traditional quality-based composition techniques usually consider the qualities at the time of the composition [6]. This is fundamentally different in cloud environments where the cloud service composition should last for a long period. Specifically, we identify the following problems in existing solutions: First, end users in cloud environment are usually large companies and organizations who aim to construct long-term business relationships with cloud service providers [4]. This aspect is largely lacking in existing service composition solutions, e.g., [5] [6]. Besides, end users and service providers participant in service composition according to their economic models [7]. However, existing models addressing economic aspects do not consider service composition but mostly focus on resource provision to specific applications [8] (e.g. cloud cache, scientific applications).

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 111–126, 2012.

This paper presents a novel quality-based cloud service composition approach. The research focuses on the selection of composition plans based solely on non-functional (Quality-of-Service, or QoS) attributes. Our main contributions include: (1) Economic models are constructed for end users to model their long-term behaviors. (2) The cloud service composition problem is considered from a decision analysis perspective. Specifically, this research proposes to use *Influence Diagrams* [9] to represent and solve cloud service composition problem. (3) An exemplary scenario is considered where the composition framework aids a department in a university compose cloud services to process tenure cases. Analytical and simulational results are presented to show the performance of the proposed approach.

The remainder of the paper is structured as follows: Section 2 presents a motivating scenario. Section 3 provides an overview of the cloud service composition problem. Section 4 gives a detail analysis of the research challenges and then elaborates the proposed composition approach. Section 5 evaluates the proposed approaches and shows the experiment results. Related work are presented in section 6. Section 7 concludes this paper and highlights some future work.

2 Motivating Scenario

We use the tenure process [10] in the US to motivate and illustrate the cloud service composition problem. American universities take great care in making tenure decisions. A junior professor is usually not promoted to a tenured position without demonstrating a strong record of research and teaching. Specifically, tenure decisions are made based mainly on the evaluation of a candidate's publication and citation records. A university typically includes dozens of colleges. Each college includes dozens of departments. Each department deals with multiple (most likely 5 to 10) tenure cases per year. The tenure process is highly labor intensive. The whole process is usually error prone and conducted manually. To overcome these problems, universities tend to outsource the tenure tasks (e.g., analysis, storage, computation) to clouds.

Let us consider a simple example, where University A contains only one college and the college contains only one department. Suppose the university outsources three main tasks to the clouds during 2012 and 2015. The proposed composition framework would generate a composite tenure application for University A. Specifically, the tenure application (Fig. 1) has three abstract SaaS. Tenure application will first search and find the publication and citation records of a candidate (task 1, T_1). It will then find the the publication and citation records of the comparable professors (task 2, T_2). Finally, the tenure application will generate the evaluation report (task 3, T_3). Besides these abstract SaaS, the composite tenure application also needs CPU, network and storage resources from IaaS providers. CPU services (denoted as CPU) are used to do computations on data. Storage services (denoted as Sto) are used to keep intermediate data. The whole tenure application should be as fair as and as transparent as possible. Therefore, all the input and output data, should be stored in case some

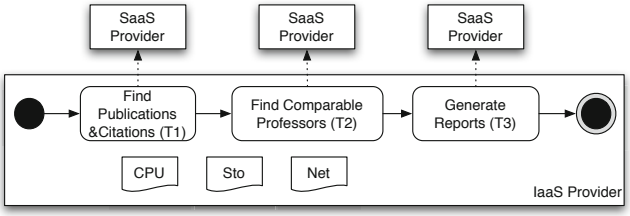


Fig. 1. Tenure application for University A

appeals arise. Network services (denoted as Net) are needed to transfer data between end users and the application, and between components in the composite application.

University A would have different QoS requirements (response time, cost etc.) on the composite tenure application during the long period, i.e., from 2012 to 2015. University A presents these preferences through a *Score Function* [6]. Composite services with higher score are more preferred. University A changes the preferences by change the weights in the score function for different periods. For example, in 2012, University A may prefer composite service that has less response time. While in 2013, University A may find response time is less important but want to save the cost as much as possible. To obtain an optimal composition, the composition framework needs a long-term economically driven model to model the preferences of the end users. Based on the user's economic model, the composition framework makes decisions to select concrete SaaS providers and IaaS providers for university A. The ultimate goal of the composition is to find an optimal plan during a long period, which has the maximal score.

3 Background

This section presents the background of the cloud service composition problem. First, the cloud environment is presented followed by the composition procedure. The adopted QoS model is then explained. Cloud service composition problem is defined at the end of this section.

3.1 Cloud Service Composition Framework

In this research, we identify four actors in the cloud environment (Fig. 2): *End Users*, *Composer*, *SaaS (Software as a Service) Providers* and *IaaS (Infrastructure as a Service) Providers*. Platform as a Service (PaaS) layer is omitted as we assume that it is included in the IaaS layer. *End Users* are usually large companies and organizations, e.g., universities, governments. The composer in this paper represents the proposed composition framework. SaaS providers supply SaaS [4] to end users. IaaS providers supply IaaS [4], i.e., CPU services, storage services, and network services, to SaaS providers and end users. The composer

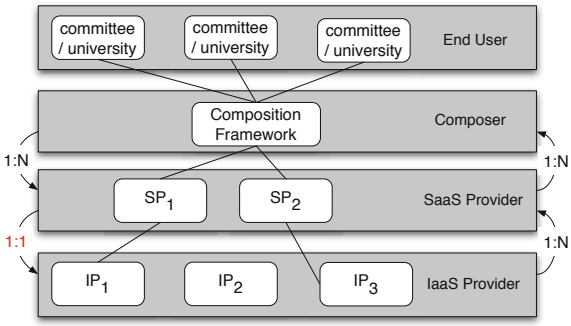


Fig. 2. Four actors in cloud computing

acts on the behave of the end users to form composite services that contains services from multiple SaaS providers and IaaS providers (Fig. 2). Here, we make the assumption that the composer interact directly with SaaS providers, SaaS providers interact directly with IaaS providers. The composer can interact with IaaS providers indirectly only through SaaSs. This assumption is reasonable since even if the composer aims to use some CPU/Network/storage resources, it must invoke these resources through some kind of software interfaces.

Similar to traditional service composition [11], cloud service composition is conducted in two steps. First, a composition schema is constructed for a composition request. Second, the optimal composition plan is selected. A composition plan is formed by choosing concrete cloud service providers for each abstract SaaS and abstract IaaS in the composition schema. Since the research focuses on the selection of composition plans based solely on QoS attributes, we assume that existing composition techniques for matching functional attributes will be used, e.g., [5] to generate composition schema.

A *Composition Schema* (or *Abstract Composite Service*) is constructed using *abstract SaaS* and *abstract IaaS*, and combined according to a set of *composition patterns*. There are four *Composition Patterns* according to the data-flow and control-flow: *S*equential Pattern (SP), *P*arallel Pattern (PP), *O*ptional Pattern (OP) and *L*oop Pattern (LP) [12]. To simplify the discussion, we initially assume that all the abstract composite services we deal with are acyclic. If an abstract composite service contains cycles (LP), a technique [6] for unfolding it into an acyclic composition schema will be applied. Composition schema is represented using Directed Acyclic Graph (DAG). Ovals denote abstract SaaS. Rectangles denote abstract IaaS. Arcs among nodes (i.e., ovals and rectangles) represent the control flow and data flow. To differentiate PP from CP, we use normal lines to represent PP and dotted lines to represent CP. A composition schema may have multiple *Execution Paths*[6] if the schema contains CP patterns. For example, in the motivating scenario, the composition schema for the tenure requests is presented in Fig. 3. T_1, T_2, T_3 are abstract SaaSs. CPU_i denotes the computation request for the output data from T_i. For example, the composite application

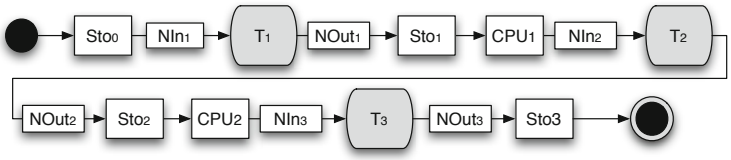


Fig. 3. Composition schema for University A

needs CPUs after receives output data from T_1. These CPUs adapt the output data to the input of T_2. Sto_i denotes the storage request for the intermediate data $NOut)i$. All the intermediate data are stored in case someone will appeal the decisions in the following years. NIn_i denotes the network resources for inputs, $NOut_i$ represents the network resources for outputs.

3.2 QoS Model

To differentiate composition plans during selection, their non-functional properties need to be considered. For this purpose, we adopt a discrete QoS model that is applicable to all the SaaS and IaaS. Without loss of generality, we only consider the QoS attributes listed as follows. Although the adopted QoS models have a limited number of attributes, they are extensible and new QoS attributes can be added. We assume IaaS are homogeneous. One unit of IaaS, i.e., *CPU, network* or *storage*, possess the same resources.

QoS Model for Elementary Services. Three QoS attributes are considered for component services: throughput, response time, and cost.

- Throughput. Given an SaaS provider SP, the throughput of its SaaS $q_{sr}(SP)$ is the number of requests the SaaS provider is able to process per second. Given an IaaS provider IP, the service rate of its IaaS $\overrightarrow{q_{sr}(IP)} = [q_{sr}^{CPU}(IP), q_{sr}^{Net}(IP), q_{sr}^{Sto}(IP)]$ is a three-attribute vector, where $q_{sr}^{CPU}(IP)$ ($q_{sr}^{Net}(IP)$, $q_{sr}^{Sto}(IP)$) represents the number of CPU (network, storage) requests the IaaS provider is able to process per second.
- Response time. Given an SaaS provider SP, the response time of its SaaS $q_{rt}(SP)$ measures the expected delay in seconds between the moment when a request is sent and the moment when the results are received. Given an IaaS provider IP, the capability of its IaaS $\overrightarrow{q_{cap}(IP)} = [q_{cap}^{CPU}(IP), q_{cap}^{Net}(IP), q_{cap}^{Sto}(IP)]$ is a three-attribute vector, where $q_{cap}^{CPU}(IP)$ ($q_{cap}^{Net}(IP)$, $q_{cap}^{Sto}(IP)$) is the number of CPU (network, storage) units used for processing a computation (data transfer, storage) request. For CPU request, the response time to adapt the output data from task t_i is calculated as: $q_{rt}^{CPU}(t_i) = \frac{CPU_i}{q_{cap}^{CPU}(IP)}$. For network request, the response time of transferring input data for task t_i is denoted as: $q_{rt}^{IN}(t_i) = \frac{NIn_i}{q_{cap}^{Net}(IP)}$. The response time of transferring output

QoS Attribute	Aggregation Functions
Throughput (q_{sr})	$q_{sr}(aCS) = min(q_{sr}(SP_1), q_{sr}(SP_2), \ldots, q_{sr}(SP_n), q_{sr}^{CPU}(IP), q_{sr}^{Net}(IP), q_{sr}^{Sto}(IP))$
Response time (q_{rt})	$q_{rt}(aCS) = CPA(aCS)$
Cost (q_{cost})	$q_{cost}(aCS) = \sum_{i=0}^{n}(q_{cost}(SP_i) + q_{cost}^{CPU}(t_i) + q_{cost}^{IN}(t_i) + q_{cost}^{OUT}(t_i) + q_{cost}^{Sto}(Sto_i)) \cdot q_{sr}(aCS)$

Fig. 4. Aggregation functions for computing the QoS of a composite service

data for task t_i is denoted as: $q_{rt}^{OUT}(t_i) = \frac{NOut_i}{q_{cap}^{Net}(IP)}$. For storage request, no response time is needed to compute, since we do not consider setup time or other time for storage resources in this research.

– Cost. Given an SaaS provider, the execution cost $q_{cost}(SP)$ is the fee that a customer needs to pay for a single request. If the SaaS provider agrees to supply SaaS at service rate $q_{sr}(SP)$. The total execution cost is computed using the expression: $cost = q_{sr}(SP) \cdot q_{cost}(SP)$. Given an IaaS provider IP, the cost for using unit IaaS for one second is denoted as a three-attribute vector $\overrightarrow{q_{cost}(IP)} = [q_{cost}^{CPU}(IP), q_{cost}^{Net}(IP), q_{cost}^{Sto}(IP)]$, where $q_{cost}^{CPU}(IP)$, $q_{cost}^{Net}(IP)$ and $q_{cost}^{Sto}(IP)$ are the price for using unit CPU IaaS, unit network IaaS and unit storage IaaS for one second correspondingly. For CPU request, the cost of computing output data from task t_i is represented as: $q_{cost}^{CPU}(t_i) = q_{cost}^{CPU}(IP) \cdot q_{cap}^{CPU}(IP) \cdot q_{rt}^{CPU}(t_i)$. The cost to transfer input data for task t_i is calculated using: $q_{cost}^{IN}(t_i) = q_{cost}^{Net}(IP) \cdot q_{cap}^{Net}(IP) \cdot q_{rt}^{IN}(t_i)$. The cost to transfer output data for task t_i can be calculated as: $q_{cost}^{OUT}(t_i) = q_{cost}^{Net}(IP) \cdot q_{cap}^{Net}(IP) \cdot q_{rt}^{OUT}(t_i)$. The cost to store intermediate data Sto_i is computed as: $q_{cost}^{STO}(Sto_i) = q_{cost}^{Sto}(IP) \cdot Sto_i \cdot time$, where $time$ denotes the seconds the intermediate data should be stored.

QoS Model for Composite Services. The quality criteria defined above are in the context of elementary cloud services. Aggregation functions are used to compute the QoS of the composite service. Fig. 4 presents these aggregation functions:

– Throughput. The throughput of a composite service denotes the number of requests it serves per second. For an abstract composite service aCS, the throughput $q_{sr}(aCS)$ is the minimal service rate of the selected SaaS providers $q_{sr}(SP)$ and the IaaS provider $q_{sr}(IP)$.
– Response time. The response time $q_{rt}(aCS)$ of an abstract composite service aCS is computed using the Critical Path Algorithm (CPA) [13]. Specifically, the CPA is applied to the execution path $Path(aCS)$ of the abstract composite service aCS. The critical path is a path from the initial node to the final node which has the longest total sum of weights labeling its nodes. In the case at hand, a node corresponds to an abstract SaaS or IaaS in an execution path, and its weight is the response time of the SaaS or IaaS.
– Cost. The cost of an abstract service is the sum of execution cost of all the selected SaaS and IaaS.

3.3 Problem Definition

Based on the analysis above, this section presents the general definition of the cloud service composition problem. Suppose an end user has a set of requests during a long period. Each request demands the same execution path (denoted as $Path = \{Sto_0, NIn_1, t_1, NOut_1, Sto_1, CPU_1, NIn_2, t_2, NOut_2, Sto_2, CPU_2, \ldots, NIn_{n_j}, t_{n_j}, NOut_{n_j}, Sto_n\}$). The end user represents its QoS preferences by determining the weights in the score function during a long period. We denote the QoS requirements of the end user as: $W(Path) = \{W(1), W(2), W(3), \ldots, W(t)\}$. Each tuple $W(t)$ represents the weights for different QoS attributes for the composite service at period $period_t$. To illustrate the composition problem, we use the three QoS attributes for saaS discussed earlier, other QoS attributes can be used instead without any fundamental changes. The QoS dimensions are numbered from 1 to 3, with 1 = throughput, 2 = response time and 3 = cost. Hence, Each $W(t)$ is further denoted as a matrix: $W(t) = [w_1(t), w_2(t), w_3(t)]$, where $w_a(t)$ denotes the weight of QoS attribute a for the composite service at period $period_t$.

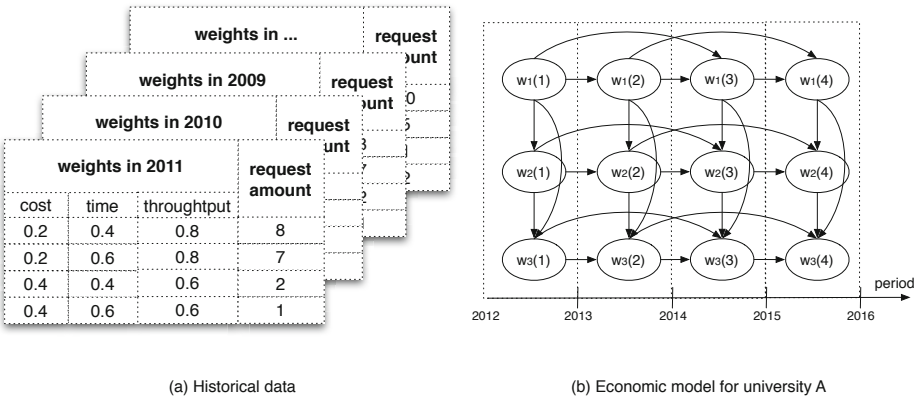
For task T_i, a set of k_i candidate SaaS providers can be used to implement the task: $SP_i = \{SP_i(1), SP_i(2), \ldots, SP_i(k_i)\}$. A set of p_p candidate IaaS providers supply IaaS to composite services: $SP_0 = SP_0(1), SP_0(2), \ldots, SP_0(p_p)\}$. A candidate composition plan (denoted as $Plan[SP_0(k_0), SP_1(k_1), SP_2(k_2), \ldots, SP_n(k_n)]$) is formed by selecting certain SaaS providers and IaaS providers for an end user. In the composition plan, the composite service is supported by the IaaS provider $SP_0(k_0)$. Task T_i is implemented by SaaS provider $SP_i(k_i)$. The QoS values for a composition plan $Plan$ is denoted as: $q(plan) = \{q(1), q(2), q(3), \ldots, q(t)\}$. Each tuple $(q(t))$ is further denoted as a matrix: $[q_1^0(t), q_1^1(t), \ldots, q_1^i(t), q_2^0(t), q_2^1(t), \ldots, q_2^i(t), q_3^0(t), q_3^1(t), \ldots, q_3^i(t)]$, where $q_a^i(t)$ denotes the advertising QoS value of attribute a for the abstract SaaS T_i at period $period_t$. Each composite plan has an aggregated QoS values computed using the aggregation functions stated above. These values are then scaled using the SAW method in [6] to a real number in $[0, 1]$. $q_a^t(Plan)$ denotes the scaled value of QoS attribute a for the composition plan at period $period_t$. Each composition plan is associated with a "score" from the end user's perspective. A commonly used score function is the weighted sum of QoS values of the composite service:

$$S^t(Plan) = w_1(t) \cdot q_1^t(Plan) + w_2(t) \cdot q_2^t(Plan) + w_3(t) \cdot q_3^t(Plan), \tag{1}$$

The score function over the long period is then represented as: $S(Plan) = \int_1^t s^t(Plan)dt$. The composition problem is to find an optimal composition plan, which has the maximal score value.

4 ID-Based Composition Approach

This section first presents the economic model for end users. Influence diagram approach is then detailed. The proposed composition approach is presented at the end of this section.



(a) Historical data

(b) Economic model for university A

Fig. 5. Economic model: an example

4.1 Cloud Economic Model

When the composition system makes decisions on which concrete SaaS providers and IaaS providers should be selected for the end user, it has no idea about how will the ultimate composite service behave during a long period. To enable long-term cloud service composition, economic models are needed to predict the long-term preferences of the end users. An economic model is defined as "a theoretical construct that represents economic processes by a set of variables and a set of logical and quantitative relationships between them. "[14].

We adopt Bayesian Networks (BN) [15] to represent the economic model for the end users. BN is a probabilistic graphical model that represents a set of random variables and their conditional dependency using a directed acyclic graph. A BN consists of a set of random variables as nodes which are connected through directed links (arcs). Each node has a conditional probability table that quantifies the effects the parents have on the nodes. If we represent the weights at different periods as nodes in a BN, we can then leverage the network as a means to represent the economic model for the end users.

For end users, we make the assumption that all the requests, initialised at the same period, have the score function with the same weights. However, requests initialised at different periods have different score functions. For example, Fig. 5 shows the economic model of university A in the tenure example. This economic model is constructed based on historical data (Fig. 5(a)) from university A. Fig. 5(a) shows the QoS preferences of university A for the last several years. The weight for each QoS attribute is a real number between 0 and 1. The larger the weight the more important is the corresponding QoS attribute to university A. The last column represents the number of tenure requests that have the preferences. Based on these historical data, we construct the economic model for university A as shown in Fig. 5(b). $w_a(t), a = 1, 2$ denotes the weight for QoS attribute a at period $period_t$. In the same period $period_t$, the weights have

conditional probability relationship with each other, i.e., $w_2(t)$ depends on the value of $w_1(t)$. For different periods, a weight $w_a(t)$ at period $period_t$ would depend on the weights in the previous periods, i.e., $w_a(1)$, $w_a(2)$, ..., $w_a(t-1)$. However, this research only considers two previous weights, since it is reasonable to assume that the weights at the most recent periods will have more affection on the weights at present. Hence, as shown in Fig. 5(a), $w_a(t)$ depends on the values of $w_a(t-1)$ and $w_a(t-2)$.

4.2 Influence Diagram Problem

Based on the economic models for the end users, we adopt *Influence Diagram* to represent and solve the cloud service composition problem. Influence diagrams (IDs) [9] are graphical models for representing and solving complex decision-making problems based on uncertain information. IDs are directed acyclic graphs that is seen as BN augmented with decision and value nodes.

An ID is a directed acyclic graph (N, A): $N = D \cup C \cup U$. D correspond to a set of decision variables under the control of the decision maker. C is the set of chance nodes correspond to random variables. U is a set of utility nodes that represent the objective functions of the model. A is the set of directed arcs between the nodes. Arcs pointing to a decision node indicate what information will be known to the decision maker at the time the decision is made. Arcs to a chance node indicate which variables condition the probability distribution for the associated random variables. Arcs to a utility node indicate which variables condition the associated expected utility. Each node in an ID is associated with a frame of data. For a chance node x, this data includes the outcome space of x, Ω_x, and the conditional probability distribution of x, π_x. For each decision node d, this data includes the alternatives of the associated decision variable, Ω_d. Finally, the data frame for utility node r contains the conditional expected value of r conditioned on the predecessors of r. The conditional expectation of r is actually a deterministic function of the conditioning variables: $U[r|C(r)] = g(C(r))$. The outcome space of r is Ω_r. To solve an ID problem (or to evaluate an ID problem) is to find the maximal utility value and the decision values at the time when the utility values are maximised.

Regarding cloud service composition problem stated above, we model cloud service composition problem to an ID problem as follows. We represent the weights from end users and the advertising QoS values from cloud providers as chance nodes in an ID.The QoS values from cloud service providers have conditional probabilistic relationship. All the selection decisions on abstract SaaS and IaaS are represented as decision nodes in the ID. Utility nodes represent the score of a composition plan. For example, Fig. 6 shows the influence diagram representation for the tenure example. In Fig. 6, node $q_a^i(t)$ in denotes the advertising value of QoS attribute a for task T_i at period $period_t$. $wq_a(t)$ denotes the weighted score for QoS attribute a at period $period_t$. $S(t) = wq_1(t) + wq_2(t) + wq_3(t)$, denotes the score for the composition plan at period $period_t$, which can be computed using Equ. 1. $S_{total} = S(1) + S(2) + S(3)$, denotes the total score

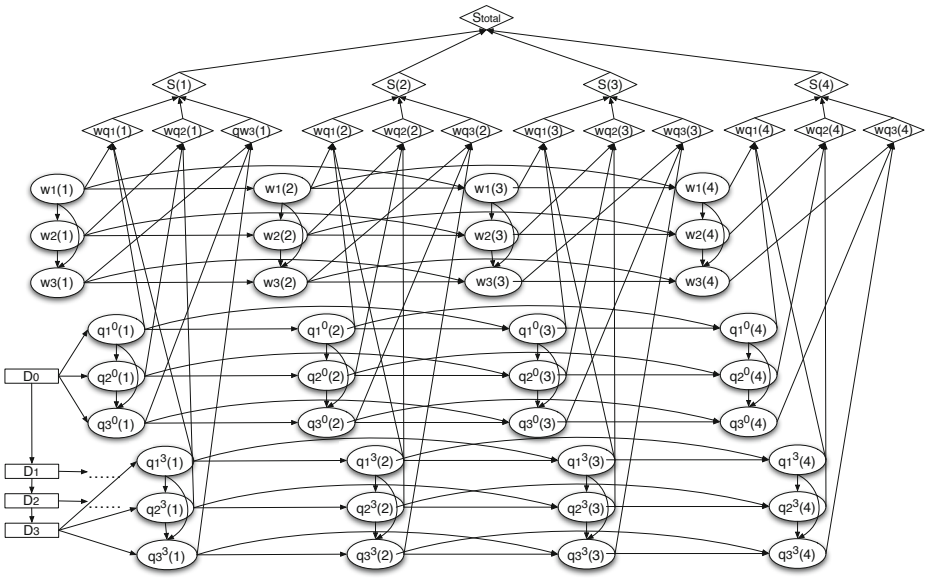


Fig. 6. Influence diagram for the tenure example when one QoS attribute is considered

for this ID. The separable nature of the utility function is represented in the structure of the graph using these multiple utility nodes. In the cloud service composition problem, there are two kinds of utility nodes. A *super utility node* is either a sum node or product node (e.g., $S(1)$, $S(2)$, S_{total}), and a *non-super utility node* is any other utility node (e.g., $wq_1(1)$, $wq_2(1)$). There is exactly one utility node, the *terminal utility node* (e.g., S_{total}), which has no successors in an ID. This represents the objective function for the model. Super utility nodes can only have utility nodes (either super or non-super) as conditional predecessors. Non-super utility nodes, on the other hand, can only have chance and decision nodes as conditional predecessors.

4.3 Dynamic Programming Algorithm

ID problems can be solved using two types of solutions [16]: A brute force solution is first transfer the ID to the corresponding decision tree, then compute all the possible scenarios with their probability and finally obtains the optimal decision variables that maximise the utility value. Another type of solution is to iteratively reduce the diagram using influence diagram reductions [17]. Considering the properties of cloud service composition, we propose a dynamic programming reduction algorithm to solve the ID problem.

Solving an ID problem using reductions involves applying a sequence of maximization and expectation operators to the utility function. In the influence diagram, these operators correspond to remove decision and chance nodes at the

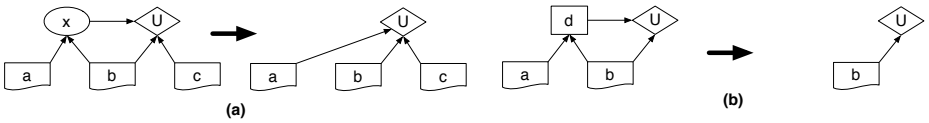


Fig. 7. Influence diagram reductions

utility node by performing maximizations or expectations. The special properties of the maximization and expectation operators when applied to separable functions are foundation of the proposed dynamic programming algorithm. They allow maximizations and expectations to be performed over an addend or factor in the utility function instead of over the entire utility function. In those cases, only a subspace of the utility function needs to be examined. This may significantly reduce the dimensionality of the operations necessary to solve a decision problem.

Two main reductions to solve an ID problem are: removing a chance node by expectation and removing a decision node by maximization. Removal of a chance node by expectation, as in Fig. 7(a), can be performed whenever the only successor of a chance node is the utility node. In the mathematics this corresponds to: $Utility[U|a,b,c] \leftarrow Utility_{\Omega_x}[Utility[U|x,b,c]|a,b]$. Note that in this case, the utility node U inherits the predecessors of chance node x. The removal of a decision node by maximization, as in Fig. 7(b), can be performed whenever the decision node has the utility node as its only successor and all conditional predecessors of that utility node, other than the decision node, are informational predecessors of the decision node. In the mathematics, decision node removal corresponds to: $Utility[U|b] \leftarrow max_d\{Utility\ [U|b,d]\}$ and $d^* = argmax_d\{Utility[U|b,d]\}$. Note that the utility node U does not inherit any predecessors of d as a result of this operation. Consider the motivating example in Fig. 6, the cloud service composition problem can always be solved by reducing the chance nodes and decision nodes in the following sequence: $q_a^i(t)$, $w_a(t)$, D_i. Nodes $q_a^i(t)$ can be removed using expectation operators as in Fig. 7(a). Nodes D_i can be removed using maximisation operators as in Fig. 7(b). Interested readers can refer to [17] for the details of other basic reductions of ID: e.g., arc reversal using Bayes theorem, summing a variable out of the joint.

Algorithm. 1 presents the dynamic programming algorithm to solve the composition problem as an ID problem. The algorithm will continue reducing nodes from the ID until there is only one terminal utility node left (line 2). If two utility nodes r_1 and r_2 have the same successor, a super value node r, and $C(r_1)$ is contained in $C(r_2)$, then removing r_1 and r_2 (if they are the only predecessors of r, or merging r_1 and r_2, into new value node r' if they are not) will not increase the size of any operation necessary to solve the influence diagram and so we should remove them (line 3:4). After each step of the algorithm (line 6:11), the net change in total number of nodes in the diagram will be at least one less. The algorithm always reduces an influence diagram to the terminal value node thus producing the optimal policy and maximum expected value for the problem.

Algorithm 1. Dynamic programming approach to evaluate ID

```
ID ← The influence diagram with the terminal utility node U
while C(U) ≠ ∅ do
    if there is removable utility nodes then
        remove all the necessary utility nodes
    else
        if there is a removable decision node d then
            remove d and the necessary utility nodes
        else
            there must be a removable chance node x
            remove x and the necessary utility nodes
        end if
    end if
end while
```

5 Experiments and Results

We conduct a set of experiments to assess the performance of the proposed approach. We use the tenure scenario as our testing environment to setup the experiment parameters. We run our experiments on a Macbook Pro with 2.2 GHz Intel Core i7 processor and 4G Ram under Mac OS X 10.7.3. Since there is not any sizable cloud service test case that is in the tenure application domain and that can be used for experimentation purposes, we focus on evaluating the proposed approach using synthetic cloud services.

We compare the proposed approach with the brute force ID approach. The brute force approach is to generate all the possible candidate composition plans, consider all the possible scenarios for each plan regarding the economic models and compute the score and the probability for each scenario (i.e., transfer ID to a corresponding decision tree). The optimal composition plan is the one that has the maximal weighted sum score. We implement the brute force approach in Java. The dynamic programming approach is implemented using Elvira [18] and Java. Computation times are measured in experiments to compare the two approaches. In this process, the number of alternatives for each chance node and decision node is varied from 2 to 10 with the step of 1 and the length of the considered periods is varied from 2 to 5 with steps of 1. All experiments are conducted 5 times and the average results are computed.

Fig. 8 presents the computation time when the number of alternatives of the decision nodes is varied from 2 to 10. For these experiments, we set the other parameters as follows: the number of the decision nodes is set to be 4. The number of alternatives of chance nodes ($\Omega_{w_a(t)}$ and $\Omega_{q_a^i(t)}$) is set to be 2. This means there are two options for all the chance nodes in the ID. The considered period is set to be $t = 2$, i.e., we consider the tenure example for the period during 2012 and 2013. And the number of QoS attributes is set to be 3. From Fig. 8, we can see that both approaches will have polynomial time complexity. But the dynamic programming has better performance than the brute force approach.

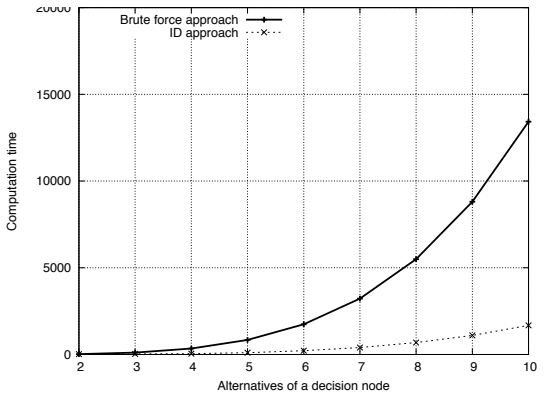


Fig. 8. Computation time VS. Ω_d

When the number of alternatives of the decision nodes is small (i.e., 2), both approaches have similar computation time. When the number of alternatives becomes larger, the dynamic programming approach behaves much better than the brute force approach.

Fig. 9 presents the computation time when the number of alternatives of the chance nodes is varied from 2 to 10. For these experiments, we set the other parameters as follows: the number of the decision nodes is set to be 4. The number of alternatives of decision nodes (D_i) is set to be 4. This means there are four options for all the decision nodes in the ID. The considered period is set to be $t = 2$, i.e., we consider the tenure example for the period during 2012 and 2013. And the number of QoS attributes is set to be 3. From Fig. 9, we can see that both approaches will have polynomial time complexity. But the dynamic programming approach behaves much better than the brute force approach.

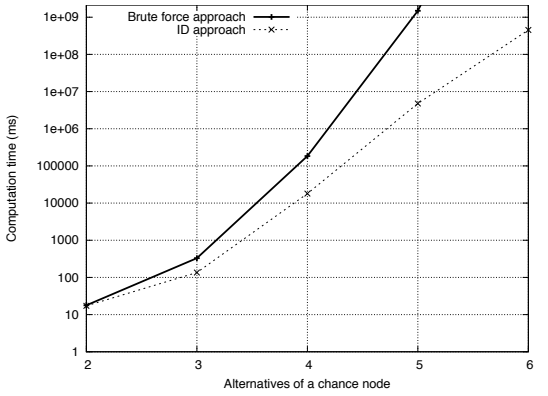


Fig. 9. Computation time VS. Ω_c

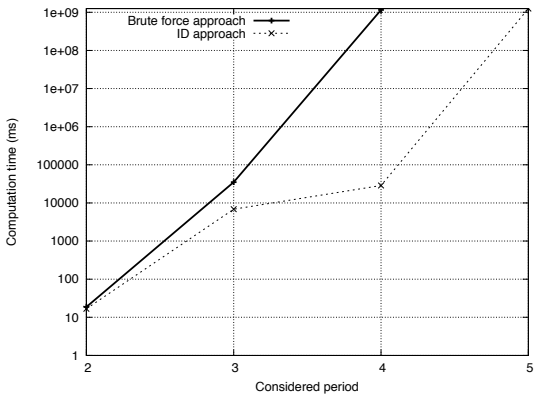


Fig. 10. Computation time VS. t

Fig. 10 presents the computation time when the considered period is varied from 2 to 5. For these experiments, we set the other parameters as follows: the number of the decision nodes is set to be 4. The number of alternatives of decision nodes (D_i) is set to be 4. The number of alternatives of chance nodes ($\Omega_{w_a(t)}$ and $\Omega_{q_a^i(t)}$) is set to be 2. This means there are two options for all the decision nodes and chance nodes in the ID. And the number of QoS attributes is set to be 3. From Fig. 10, we can see that both approaches will have exponential time complexity. When the considered period is short (i.e., 2 years or 3 years), both approaches have similar computation time. When the consider period becomes longer, the dynamic programming approach behaves much better than the brute force approach.

6 Related Work

Service composition is an active research area in service-oriented computing [5]. During the last decade, service composition problem can be categorized into two groups. One group focuses on the functional composability among component services. The other group aims to make optimal decisions to select the best component services based on non-functional properties (QoS).

Functional-driven service composition approaches typically adopt semantic descriptions of services. Examples of automatic approaches include Policy-based approach proposed by [19] and composability model driven approach proposed by [5]. Other functional-driven composition approaches use AI planning methods. Most of them [20] assume that each service is an action which alters the state of the world as a result of its execution. The inputs and outputs parameters of a service act as preconditions and effects in the planning context. Users only need to specify the inputs and the outputs of the desired composite service, a plan (or a composite service) would automatically generated by the AI planners.

Functional-driven service composition approaches mostly do not attempt to find an optimal solution but only to find a solution. However, the non-functional

properties (QoS) of resulting composite service is a determinant factor to ensure customer satisfaction. Different users may have different requirements and preferences regarding QoS. Therefore, QoS-aware composition approaches are needed. QoS-aware service composition problem is usually modelled as a Multiple Criteria Decision Making [6] problem. The most popular approaches include integer linear programming and genetic algorithms. An Integer Linear Program (ILP) consists of a set of variables, a set of linear constraints and a linear objective function. After having translated the composition problem into this formalism, specific solver software such as LPSolve [21] can be used. [22] and [23] use Genetic Algorithms (GA) for service composition. Individuals of the population correspond to different composition solutions, their genes to the abstract component services and the possible gene values to the available real services. While GAs do not guarantee to find the optimal solution, they can be more efficient than ILP-based methods (which have exponential worst-case time complexity).

Most of the existing composition approaches are not well suited for cloud environment [23]. They usually consider the qualities at the time of the composition [5]. The proposed composition approach consider the problem from a long-term perspective.

7 Conclusion

This paper proposes a cloud service composition approach to aid end users selecting and composing SaaS providers and IaaS providers in the cloud environment. Compared to traditional service composition framework in SOC, the proposed approach considers service composition from a long-term perspective. Cloud economic models for both end users and cloud service providers are leveraged during the composition. Specially, an influence diagram approach is adopted to solve cloud service composition problems. In future work, discrete QoS model will be extended to continuous model, where each chance node in an ID has infinite alternatives. Besides, machine learning algorithms will be researched on refining the economic model for both end users and cloud service providers to improve the performance.

References

1. Motahari-Nezhad, H., Stephenson, B., Singhal, S.: Outsourcing business to cloud computing services: Opportunities and challenges. IEEE Internet Computing (2009)
2. Youseff, L., Butrico, M., Da Silva, D.: Toward a unified ontology of cloud computing. In: Grid Computing Environments Workshop (2009)
3. Yu, Q., Liu, X., Bouguettaya, A., Medjahed, B.: Deploying and managing web services: issues, solutions, and directions. The VLDB Journal 17(3), 537–572 (2008)
4. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: Above the clouds: A berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28 (2009)

5. Medjahed, B., Bouguettaya, A., Elmagarmid, A.: Composing web services on the semantic web. The VLDB Journal 12(4), 333–351 (2003)
6. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)
7. Dash, D., Kantere, V., Ailamaki, A.: An economic model for self-tuned cloud caching. In: IEEE 25th International Conference on Data Engineering, pp. 1687–1693 (2009)
8. Kantere, V., Dash, D., Francois, G., Kyriakopoulou, S., Ailamaki, A.: Optimal Service Pricing for a Cloud Cache. IEEE Transactions on Knowledge and Data Engineering (2011)
9. Shachter, R.: Probabilistic inference and influence diagrams. Operations Research, 589–604 (1988)
10. Gelmon, S., Agre-Kippenhan, S.: Promotion, tenure and the engaged scholar. AAHE Bulletin 54(5), 7–11 (2002)
11. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Computing, 51–59 (2004)
12. Wu, B., Chi, C., Chen, Z., Gu, M., Sun, J.: Workflow-based resource allocation to optimize overall performance of composite services. Future Generation Computer Systems 25(3), 199–212 (2009)
13. Pinedo, M.: Scheduling: theory, algorithms, and systems. Springer (2012)
14. Baumol, W., Blinder, A.: Economics: principles and policy. South-Western Pub. (2011)
15. Jensen, F.: An introduction to Bayesian networks, vol. 74. UCL Press, London (1996)
16. Shachter, R.: Evaluating influence diagrams. Operations Research 34(6), 871–882 (1986)
17. Tatman, J., Shachter, R.: Dynamic programming and influence diagrams. IEEE Transactions on Systems, Man and Cybernetics 20(2), 365–379 (1990)
18. Elvira, a Java implementation of influence diagram (2005), `http://www.ia.uned.es/~elvira`
19. Chun, S.A., Atluri, V., Adam, N.R.: Using semantics for policy-based web service composition. Distributed and Parallel Databases 18(1), 37–64 (2005)
20. Wu, D., Parsia, B., Sirin, E., Hendler, J., Nau, D.: Automating DAML-S Web Services Composition Using SHOP2, p. 195 (2003)
21. Berkelaar, M., Eikland, K., Notebaert, P., et al.: lpsolve: Open source (mixed-integer) linear programming system. Eindhoven U. of Technology (2004)
22. Canfora, G., Di Penta, M., Esposito, R., Villani, M.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 1069–1075 (2005)
23. Ye, Z., Zhou, X., Bouguettaya, A.: Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part II. LNCS, vol. 6588, pp. 321–334. Springer, Heidelberg (2011)