QoS-Aware Cloud Service Composition Using Time Series

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

[1] The University of Queensland, Australia
[2] Royal Melbourne Institute of Technology, Australia

Abstract. Cloud service composition is usually long term based and economically driven. We propose to use multi-dimensional Time Series to represent the economic models during composition. Cloud service composition problem is then modeled as a similarity search problem. Next, a novel correlation-based search algorithm is proposed. Finally, experiments and their 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.

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 [3]. For example, which composite service has the best performance at present? This is fundamentally different in cloud environments where the cloud service composition should last for a long period. For example, which composite cloud service performs best in the next few years, despite it may not be the best one at present? This paper presents a novel cloud service composition approach based on time series. Time series databases are prevalent in multiple research ares, e.g., multimedia, statistics etc. Many techniques [4], [5] have been proposed to effectively and efficiently analyze economic models in cloud computing [] [].

We identify four actors in the cloud environment (Fig. 1): *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 [6] to end users. IaaS providers supply IaaS [6], i.e., CPU services, storage services, and network services, to SaaS providers and end users. The composer acts on

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 9–22, 2013.

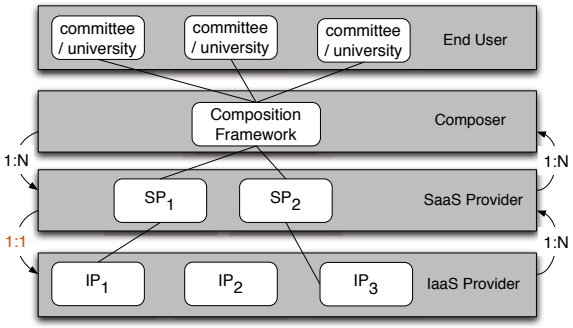

Fig. 1. Four actors in cloud computing

the behave of the end users to form composite services that contains services from multiple SaaS providers and IaaS providers (Fig. 1).

Similar to traditional service composition [7], 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.

Our research focuses on the selection of composition plans based solely on non-functional (Quality-of-Service, or QoS) attributes [8]. On one hand, we model the requirements of end users as a set of time series. On the other hand, cloud service providers market their services (SaaS or IaaS) using a set of time series. Each time series represents the values of a corresponding QoS attribute over a long period. Hence, cloud service composition problem becomes a similarity search problem whose query is a set of desired time series.

Traditional techniques seldomly handle complex time series queries which require the correlation between the time series to be used during similarity matching. The correlation, however, are prevalent during service composition where each QoS attribute is correlated with several other QoS attributes. For example, considering that we will select the best composition plan by leveraging two QoS attributes, response time and cost. The cost of a cloud service may decrease during a period, while the response time of the cloud service is also decreasing. We can easily observe the correlations that exist between time series. If we process each time series independently, we will not be able to make use of the inherent correlations.

We refer to groups of time series with correlations as Time Series Groups (TSG). Given a query TSG object $Q = \{q_1, q_2, \ldots, q_l\}$, where l is the number of the time series in Q and each time series $q_i = \{(x_m, t_m)|m = 1, 2, \ldots, L_i\}$, where x_m is the value of the time series at time t_m, and a TSG set $D = \{TSG_1, TSG_2, \ldots, TSG_N\}$. The similarity search on TSG is to find the most similar TSGs from set D via a predefined function $dist(Q, TSC_i)$, where

$$result = argmin_{i=1,\ldots,N}(dist(Q, TSC_i)). \quad (1)$$

The main contributions of the paper include: (1) Cloud service composition problem is considered as a similarity search problem. Specifically, this paper

proposes algorithms that leverages time series correlations to effectively and efficiently compute the distance between two TSGs. (2) We propose two data structures for Time Series Group processing, which leverage the principal components and the relations of time series. A key difference between TSG and existing time series similarity search is that the former requires the use of relations between time series. (3) Analytical experiments are presented to show the performance of the proposed approach. The empirical results show that the proposed approach is superior compared with other similarity search approaches.

The remainder of the paper is structured as follows: Related work are presented in section 2. Section 3 presents a use case for cloud service composition. 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. Section 6 concludes this paper and highlights some future work.

2 Related Work

Service composition is an active research topic in service-oriented computing [9]. During the last decade, many QoS-aware composition approaches are proposed. QoS-aware service composition problem is usually modelled as a Multiple Criteria Decision Making [3] 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 [10] can be used. [11] and [12] 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 traditional Web service composition approaches are not well suited for cloud environment [12]. They usually consider the qualities at the time of the composition [9]. [8] adopts Nayesian Network and Influence diagram to model the economic models and solve the cloud service composition problem. However, they assume conditional probability relationship among multiple QoS attributes, which is not a general case in reality. This paper considers a general case where QoS attributes can be described using time series.

Existing work on time series query can be categorized into two categories, time series matching and pattern and correlation mining on multiple time series. In existing similarity search over time series databases, the time series is transformed from its original form into a more compact representation. The search algorithm leverages on two steps: dimensionality reduction [4], [13], [14], [15] and data representation in the transformed space. Various dimensionality reduction techniques have been proposed for time series data transformation. These includes: Discrete Fourier Transform (DFT), Singular Value Decomposition (SVD)

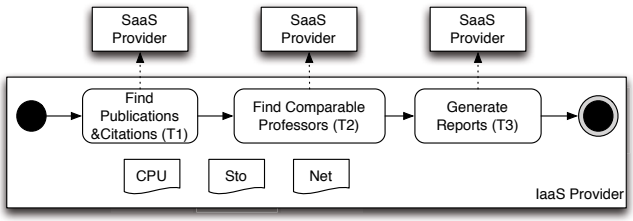

Fig. 2. Tenure application for University A

[13], Discrete Wavelet Transform (DWT)[14], and Piecewise Aggregate Approximation (PAA) [13]. Another approach for dimensionality reduction is to make use of time series segmentation [4].

Existing multiple time series research have focused on pattern mining and finding correlation between multiple time series, over patterns and observed values from group of individual time series. Papadimitriou et al. [16] proposed the SPIRIT system, which focuses on finding patterns, correlations and hidden variables in a large number of streams. SPIRIT performs incremental Principal Component Analysis (PCA) over stream data, and delivers results in real time. SPIRIT discovers the hidden variables among n input streams and automatically determines the number of hidden variables that will be used. The observed values of the hidden variables present the general pattern of multiple input series according to their distributions and correlations. These existing approaches cannot be easily extended for the TSG similarity search problem because of the lack of a clearly defined similarity measure. Most importantly, existing approaches are unable to deal with the relations that exist between the multiple time series.

3 Use Case

Let us consider a simple use case, university A requires a composite cloud service to aid the tenure process every year. Suppose the university outsources three main tasks to the clouds during 2012 and 2015. The aim of cloud service composition in this example is to find and select a set of component cloud services to form a tenure application. Specifically, the tenure application (Fig. 2) 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 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 has different QoS requirements (e.g., the number of tenure cases that the composite service can process at once, the cost of the service and the reputation of the service) on the composite tenure application over a period of time. These preferences are presented through a set of time series data. Each time series denotes the requirement of the university on a specific QoS attribute. All the cloud service providers also advertise their services using time series with the same time frequency. Hence, there are multiple candidate composition plans, which are also represented as a set of time series data by aggregating all the component services together. The composition problem, therefore, becomes a query problem that aims to find the most similar sets of time series to meet the university's requirements.

4 Time Series Group Search Approach

There are two requirements when designing our composition approach. First, a time series representation is needed to describe and measure the general information extracted from a TSG. Second, the similarity search algorithm must be more scalable compared to other approaches, since cloud environment is more scalable than other existing platforms.

This section first presents the QoS model for cloud service composition. Two data structures, denoted as QA and QR are then introduced to represent time series during cloud service composition. The proposed composition approach is finally presented at the end of this section.

4.1 QoS Model

To differentiate composition plans during selection, their non-functional properties need to be considered. For this purpose, we adopt a 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.

Three QoS attributes are considered for component services and composite services: throughput, reputation, 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.

- Reputation. Given an SaaS provider SP, the reputation $q_{rt}(SP)$ is a value between $(0, 1)$. Given an IaaS provider IP, the reputation of its IaaS $\overrightarrow{q_{rt}(IP)} = [q_{rt}^{CPU}(IP), q_{rt}^{Net}(IP), q_{rt}^{Sto}(IP)]$ is a three-attribute vector, where $q_{rt}^{CPU}(IP)$ ($q_{rt}^{Net}(IP)$, $q_{rt}^{Sto}(IP)$) is the reputation for processing a computation (data transfer, storage) request.
- Cost. Given an SaaS provider, the execution cost $q_{cost}(SP)$ is the fee that a customer needs to pay for the SaaS. Given an IaaS provider IP, the cost for using IaaS 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 CPU IaaS, unit network IaaS and unit storage IaaS correspondingly.

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.

- 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 $q_{sr}(IP)$.
- Reputation. The reputation $q_{rt}(aCS)$ of an abstract composite service aCS is computed by multiplying the selected SaaS providers and IaaS providers.
- Cost. The cost of an abstract service is the sum of execution cost of all the selected SaaS and IaaS.

4.2 QoS Attribute of TSG

One of the key challenges of performing TSG similarity search in cloud service composition is to efficiently match the QoS attribute (QA) of multiple time series in TSGs. Using a compact representation can avoid the need to perform pairwise comparisons of the time series, which is computationally expensive.

Principal Component Analysis (PCA) is commonly used in time series analysis [16] to reduce dimensionality, and to facilitate query retrieval. The essential idea in PCA is to transform a set of observed values in the high-dimensional vector space into a smaller new vector space. Given a data matrix X, and a transformation matrix W. Each column of X corresponds to a data vector in the original space. Each row of W corresponds to a transformation vector. The transformation vector transforms the data vectors from the original space to the value of certain principal components (PCs). The PCA transformation is given as: $Y = X \times W$. Y denotes the data matrix in the new vector space. In the new vector space, each dimension corresponds to the principal component of the original space. It is generated from the original dimensions based on the statistical distribution of the observed values.

We adopt PCA to produce QA, a compact representation of the patterns of multiple time series in a TSG. Each QA consists of one or several time series of the principal components extracted from the original TSG. Given a TSG, which consists of n time series $T_1, T_2, \ldots, T_n$, for each time series T_i, the observed

value in time t_j is x_{t_j}. Assume the number of time slots is denoted as l, the TSG can also be represented as a $ln-$dimensional vector. We build a n-dimensional vector $X_j = (x_{1_j}, x_{2_j}, \ldots, x_{n_j})$ for each time slot. Each dimension refers to a time series in $T_1, T_2, \ldots, T_n$. We perform PCA transformation on these n-dimensional vectors to identify the first few principal components. In the implementation, we choose the first n' principal components as the QAs ($n' \leq n$). These n' PCs can capture the most dominant information of the original data, even if $n' \ll n$. We only consider these PCs and their associated observed values which have been transformed in PCA dimensions. The new observed values of time slot t_j on the dimensions of transformed PCA space are:

$$\overline{X_j} = X_j \times W = (X_j \times W_1, X_j \times W_2, \ldots, X_j \times W_n). \tag{2}$$

where W_i is a n-dimensional transformation vector. The derived values in each time slot t_j in QA are the first n' principal components of X_j, which are the first n' dimensions of $\overline{X_j}$. The transformation matrix which transforms the origin vector space into first n' dimensions is: $W' = (W_1, W_2, \ldots, W_{n'})$. Hence, QA can be represented as:

$$QA = X \times W' = (X_1 \times W', X_2 \times W', \ldots, X_l \times W'), \tag{3}$$

where l is the length of time series. The $l \times n'$ data matrix in this equation represents n' time series which are generalized form the multiple patterns of original n time series.

Comparing to Brute-Force method (mentioned in Sec. 5), which finds the best matches between every time series pair, QA is a more general feature and is easier to measure the similarity, though information will be lost by only reserving the first few principal components. The lost information can be compensated using QoS Relation (QR) which will be introduced in the next section.

4.3 QoS Relation for TSG

This section shows how we can generate relations among time series in a TSG. Relations between multiple time series can be complex. Similar to "Relation Vector" in [17], a QoS Relation (QR) of a TSG is a multidimensional vector, which can be used to describe the relations among multiple time series in a TSG. QR consists of a set of signatures, which are extracted from a relation matrix. The relation matrix is obtained based on the relation descriptors of every time series pair in TSG.

QR can be generated in mainly two steps:

- First, a relation descriptor is used to capture the intrinsic relationship for any two time series in a TSG.
- Second, PCA transformation is used to transform the high-dimensional relation descriptors into a smaller space.

The relations between T_m and T_n refer to the differences between the QoS values in the time series and the associated time slots. Hence, we introduce two

basic relations defined by [17], i.e., variance in QoS values (VIQ) and variance in time (VIT). The VIQ is a high-dimensional vector, each dimension of which captures the difference between the sampled observed QoS values of T_m and T_n:

$$(|x_{m_{s1}} - x_{n_{s1}}|, |x_{m_{s2}} - x_{n_{s2}}|, \ldots, |x_{m_{sl}} - x_{n_{sl}}|), \tag{4}$$

where $x_{m_{si}}$ is the sampled QoS values of T_m, and $x_{n_{sj}}$ is the sampled QoS value of T_n. VIT is a descriptor of the relation between intervals of T_m and T_n denoted as a 4D vector $VIT(T_m, T_n)$ [17]:

$$VIT(T_m, T_n) = (t_{m1} - t_{n1}, t_{ml} - t_{nl}, t^3_{m,n}, t^4_{m,n}), \tag{5}$$

where t_{m1} and t_{n1} are the first time ticks of T_m and T_n, t_{ml} and t_{nl} are the last time ticks. $t^3_{m,n}$ and $t^4_{m,n}$ are used to describe the overlap status between two time series' interval. They are defined as:

$$t^3_{m,n} = \begin{cases} 0, m = n, \\ t_{m1} - t_{nl}, t_{m1} \geq t_{n1}, \\ -(t_{n1} - t_{ml}), t_{ml} < t_{n1}. \end{cases} \tag{6}$$

$$t^4_{m,n} = \begin{cases} 0, m = n, \\ t_{ml} - t_{n1}, t_{m1} \geq t_{n1}, \\ -(t_{nl} - t_{m1}), t_{ml} < t_{n1}. \end{cases} \tag{7}$$

In summary, the relation descriptor of T_m and T_n is the combination of two components, i.e., the differences between QoS values and intervals.

The relation descriptors of each time series pair in TSG are vectors in a high-dimensional vector space. Given a TSG object which has n time series $T_1, T_2, \ldots, T_n$, we can generate n^2 relation descriptors R_{ij} of each pair of time series T_i and T_j from the TSG object, where $R_{ij} = \{VIQ(T_i, T_j), VIT(T_i, T_j)\}$. Using these n^2 vectors, we perform PCA to find the Principal components according to the datas latent correlation and distribution on these dimensions. Set r as the dimensionality of each R_{ij} , and $r \times r$ transformation matrix W for PCA as $(W_1, W_2, \ldots, W_r)$, we reserve only first few PCs which contain the majority information. The number of PCs remained is denoted as r'. Thus, we obtain the transformed relation descriptors $\overline{R_{ij}}$

$$\overline{R_{ij}} = (R_{ij} \times W_1, R_{ij} \times W_2, \ldots, R_{ij} \times W_{r'}). \tag{8}$$

4.4 Similarity Search of Time Series Groups

Given two TSG objects which are represented by aforementioned features, e.g., $TSG_i = \{QA_i, QR_i\}$ and $TSG_j = \{QA_j, QR_j\}$, we define the distance function between two TSGs $dist(TSG_i, TSG_j)$ based on the similarity measure for QA, i.e., $DIS(QA_i, QA_j)$ and relation vector QR, i.e., $DIS(QR_i, QR_j)$. For simplicity

of presentation, we deploy the euclidean Distance as the default distance function. The distance between two QAs can be calculated by the following equation:

$$DIS(QA_i, QA_j) = \sqrt{\sum (qc_i^k - qc_j^k)^2} \tag{9}$$

where qc_i^k and qc_j^k are the values for the k principal components of TSG_i and TSG_j.

The distance between two QRs can be calculated:

$$DIS(QR_m, QR_n) = \sqrt{\sum (\overline{R_{ij}^m} - \overline{R_{ij}^n})^2} \tag{10}$$

According to the distance functions defined on the two features, the overall distance between two TSG objects $dist(TSC_i, TSC_j)$ can be calculated by the geometric value of the two distances:

$$dist(TSC_i, TSC_j) = \sqrt{DIS(QA_i, QA_j) \times DIS(QR_i, QR_j)}. \tag{11}$$

By using these two components for TSG similarity evaluation, we solve the two challenges of effectively evaluating relations and measuring the patterns of multiple time series in TSG; therefore, our approach can perform effectively and efficiently for similarity search in TSG databases.

5 Experiments and Results

We conduct a set of experiments to assess the performance of the proposed approach. We measure both the query effectiveness and the execution time for TSG retrieval. We compare our proposed approach named TSG with the Brute-Force approach, and the QA-only approach. QA-only approach is the same with the proposed approach in Sec.4 except that it only consider QAs when computing the distance between two TSGs. All experiments are conducted using Matlab. We run our experiments on a Macbook Pro with 2.2 GHz Intel Core i7 processor and 4G Ram under Mac OS X 10.8.3. Since there is no testbed available, we focus on evaluating the proposed approach using synthetic cloud services.

5.1 Brute-Force Approach

To show the performance of the proposed approach, we first present a Brute-Force approach for solving the TSG matching problem, which exhaustively compares the time series in TSGs. In such a Brute-Force approach for TSG similarity matching, the distances between all the QoS attributes in the query time series and the candidate plan's time series are computed. The maximum similarity (i.e., minimum distance) is then used to represent the similarity between two TSGs. Here, we computing the similarity between TSG_1 and TSG_2 by calculating the euclidean distances. The Brute-Force approach guarantees the minimum of

$$\sum_{T_i \in TSG_1 \bigwedge T_i' \in TSG_2} DIS(T_i, T_i'), \tag{12}$$

where $T_i' \in TSG_2$ and T_i in TSG_1 are QoS attributes. In summary, by enumerating all pairs between the time series in two TSGs, we can calculate the minimum distance of Brute-Force approach by the following equation:

$$dist_{BF}(TSG_1, TSG_2) = min_k(\sum(DIS(T_m, T_n))), \tag{13}$$

The Brute-Force TSG similarity matching approach is an exhaustive matching-based algorithm. The approach is very costly, as it requires an enumeration between all pairwise single time series. Meanwhile, this method can measure the patterns between time series in two TSGs well, but ignores the correlations of time series inside the TSG.

5.2 Data Description

To generate the synthetic data sets, we first randomly generate 5 time series. Each time series is denoted as $\{(QoS_i, t_i)|i = 1, 2, 3, \ldots, l\}$, where the length l is equal to 150. These five time series form the seed TSG. For each time series in the seed TSG, we add some normally distributed random noise to create another 199 TSGs. Using the 200 TSGs as seeds, we expand them to 200 TSG categories, each of which is with 80 TSGs. We produce variations of TSGs by adding small time warping (e.g., 5 percent of the series length), and some normally distributed random noise. Consequently, our synthetic data set contains 200 categories and each has 80 similar TSGs, i.e., the TSG database has 16,000 TSGs in total. We use the first 200 TSGs as the query set.

We generate two dataset to evaluate the proposed approach. Time series in *RAND* dataset are generated randomly in Matlab, while time series in *GAS* dataset are generated with Gaussian distribution. For each TSG in the query set, we search the similar TSG in the whole dataset using three approaches: TSG, QA-only and the brutal force approach. Each approach will return a sorted list of the most similar TSG to the query.

We use *recall* and *precision* to compare different algorithms. We denote the first 100 results returned by the brutal force approach as the *relevant results*. Hence, the precision and the recall can be calculated using:

$$precision = \frac{\{relevant\} \bigcap \{retrieved\}}{\{retrieved\}} \tag{14}$$

$$recall = \frac{\{relevant\} \bigcap \{retrieved\}}{\{relevant\}} \tag{15}$$

5.3 Performance

In the first experiment, we investigate two important parameters of the proposed TSG retrieval method. The first parameter is the number of PCs reserved in QA feature generation. We extract QA feature by using first n' PCs of multiple time series. The second parameter is the reduced dimensionality of Relation

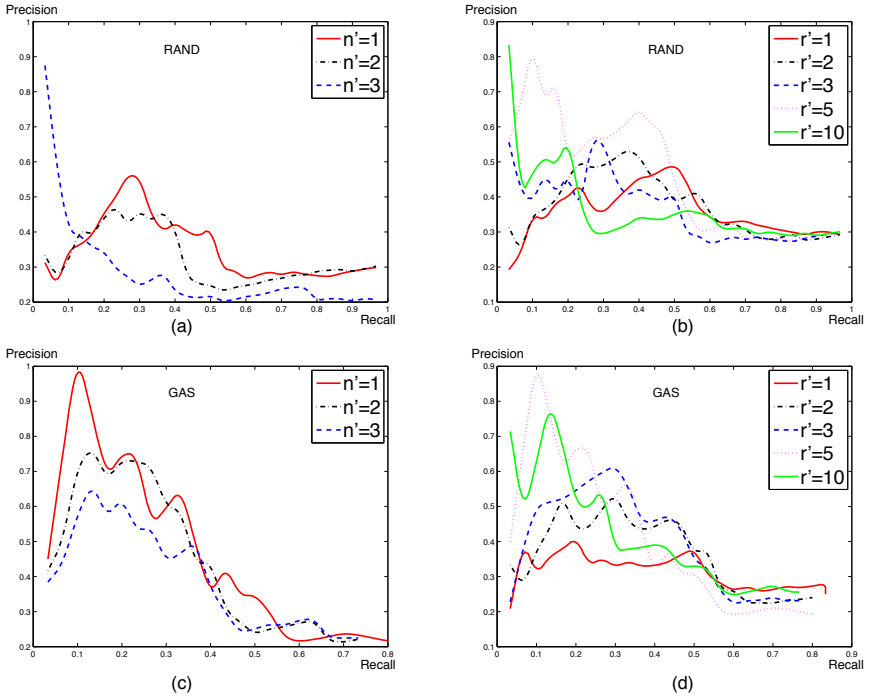

Fig. 3. Performance on parameter tuning. (a) varying the number of PCs in QA using RAND data set. (b) Varying the number of PCs in QR using RAND data set. (c) varying the number of PCs in QA using GAS data set. (b) Varying the number of PCs in QR using GAS data set.

Descriptor in QR , i.e., r', which is used to capture the relations among the time series in a TSG.

We first evaluate the effect of QA feature on TSG retrieval performance by varying the number of PCs n'. As the maximum number of PCs in QA generation is the number of time series in TSG, we conduct the experiments by varying n' from 1 to 5. The results of precision/recall for TSG retrieval are shown in Fig. 3. Each line in the graph represents the performance of a corresponding approach. Each node on a line represents the precision and the recall when the first num ($num \geq 100$) results are returned. For example, in the first graph in Fig. 3, the first node on the green line represents the precision and the recall when the first 100 results are returned.

According to Fig. 3a and Fig. 3a, the TSG approach performs best when $n' = 1$, although when it comes to RAND data set, $n' = 3$ performs better than $n' = 1$ when the recall is less than 0.16. The approach performs similarly when the recall is high. This can be explained by that when almost all the relevant results are retrieved, the precision tends to be the same. To sum up, the best performance

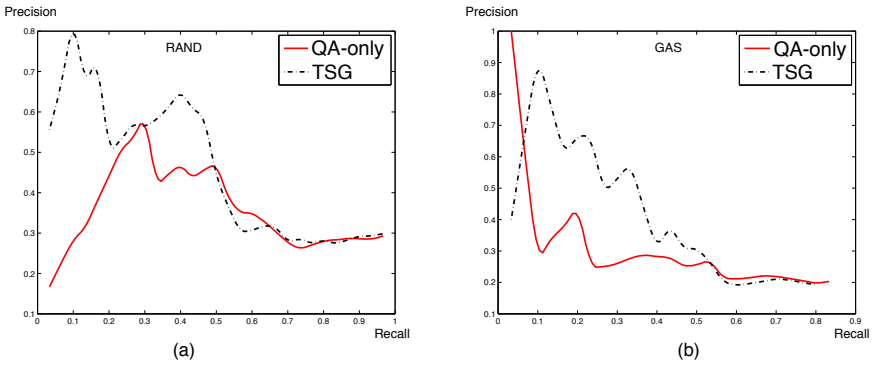

Fig. 4. Performance comparison of effectiveness: precision and recall. (a) Results using RAND data set. (b) results using GAS data set.

is obtained when only the first PC is reserved. This is because the first PC summarizes sufficient general information of multiple patterns in TSGs, while the retrieval performance by using more PCs may suffer from redundant and useless information. Note that, when $n' > 1$, we generate multiple representative time series as QA features, and hence the incurred multiple time series matching may degrade overall performance.

Next, we investigate the performance of TSG retrieval by varying the reduced dimensionality of Relation Descriptor in QR feature, i.e., r', which is shown in Fig.3b and Fig.3d. From these figures, we can observe that as more PCs are used, the performance is improved. This is because more information is included in the relation descriptor of the QR. Notice that when the recall is higher, the approach performs more similarly. Therefore, in the following experiments, we fix the two parameters for performance comparison, i.e., $n' = 1$ and $r' = 5$.

5.4 Comparison with Other Methods

In this experiment, we compare the retrieval performance of our approach with the the QA-only method. As we can see from Fig. 4, the proposed TSG approach performs similarly as the QA-only approach when the recall is bigger than 0.5 for RAND data set and 0.55 for GAS data set. This is because that when recall gets larger, more relevant results are retrieved. However, we can still observe that when the recall is small, i.e., retrieving a few most similar results, which is more common in the similarity query area, TSG still preforms better than QA-only approach. This is because the compact representation of multiple time series in QA-only approach, can only capture the limited pattern information, while ignore the intrinsic relations between the multiple time series in the TSG, which are more valuable to distinguish different TSCs. QA feature matches the similarity of group of time series by generalizing their characteristic patterns, and the QR information can capture the natural relations among time series in TSGs. The TSG approach can take both factors into effect for TSG retrieval, and hence improve the retrieval effectiveness in TSG databases.

5.5 Scalability and Robustness

In this section, we investigate the scalability and robustness of the proposed approaches. We first evaluate the time efficiency for the three approaches in terms of the scalability of the database size. We vary the size of TSG data set by utilizing 50, 100, 150, 200 categories of TSGs, which results in the data size from 4,000 to 16,000 TSGs.

Fig. 5 shows the average time cost of these different approaches for one TSG retrieval. We can observe that QA-only approach and the proposed approach are comparable and perform much better than the Brute-Force method. Our TSG approach explores both QA and QR for similarity computation, and the extra computation on QR feature is needed compared with QA-only approach. While the Brute-Force algorithm performs badly by two orders of magnitude. This is because the Brute-Force algorithm needs to calculate all the matches for each time series pair in the group.

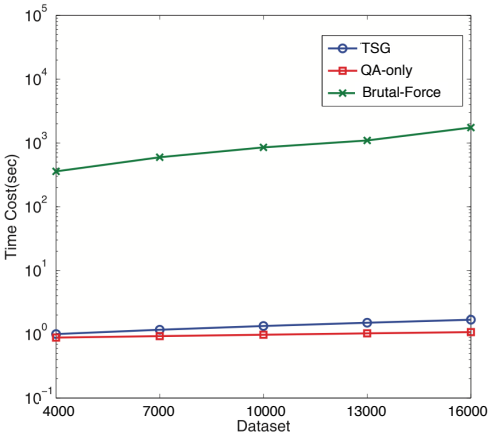

Fig. 5. Time efficiency of different approaches

6 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, we propose to use time series to represent the economic models. Cloud service composition is then modeled as a similarity search problem in the multiple time series database. We use two structure QoS attribute and QoS relation to further improve the effectiveness and the efficiency of the similarity search approach.

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. 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)
4. Keogh, E., Chu, S., Hart, D., Pazzani, M.: Segmenting time series: A survey and novel approach. Data Mining in Time Series Databases 57, 1–22 (2004)
5. Bashir, F.I., Khokhar, A.A., Schonfeld, D.: Real-time motion trajectory-based indexing and retrieval of video sequences. IEEE Transactions on Multimedia 9(1), 58–65 (2007)
6. 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)
7. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Computing, 51–59 (2004)
8. Ye, Z., Bouguettaya, A., Zhou, X.: QoS-aware cloud service composition based on economic models. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 111–126. Springer, Heidelberg (2012)
9. Medjahed, B., Bouguettaya, A., Elmagarmid, A.: Composing web services on the semantic web. The VLDB Journal 12(4), 333–351 (2003)
10. Berkelaar, M., Eikland, K., Notebaert, P., et al.: lpsolve: Open source (mixed-integer) linear programming system. Eindhoven U. of Technology (2004)
11. 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)
12. 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)
13. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. Knowledge and Information Systems 3(3), 263–286 (2001)
14. Kahveci, T., Singh, A.: Variable length queries for time series data. In: Proceedings of the 17th International Conference on Data Engineering, pp. 273–282. IEEE (2001)
15. Wu, Y.-L., Agrawal, D., El Abbadi, A.: A comparison of dft and dwt based similarity search in time-series databases. In: Proceedings of the Ninth International Conference on Information and Knowledge Management, pp. 488–495. ACM (2000)
16. Papadimitriou, S., Sun, J., Faloutsos, C.: Streaming pattern discovery in multiple time-series. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 697–708. VLDB Endowment (2005)
17. Cui, B., Zhao, Z., Tok, W.H.: A framework for similarity search of time series cliques with natural relations. IEEE Transactions on Knowledge and Data Engineering 24(3), 385–398 (2012)