AI Planning and Combinatorial Optimization for Web Service Composition in Cloud Computing

Guobing Zou[1, 2] Yixin Chen[2] Yang Xiang[1] Ruoyun Huang[2] You Xu[2]

[1]Tongji University
Department of Computer Science
4800 Cao'an Road
Shanghai 201804, China
1-314-935-8501
guobingzou@gmail.com

[2]Washington University
Department of Computer Science
Campus Box 1045
One Brookings Drive, St. Louis 63130, USA
1-314-935-7528
{chen, rh11, yx2}@cse.wustl.edu

ABSTRACT

In recent years, there has been an increasing interest in web service composition due to its importance in practical applications. At the same time, cloud computing is gradually evolving as a widely used computing platform where many different web services are published and available in cloud data centers. The issue is that traditional service composition methods mainly focus on how to find service composition sequence in a single cloud, but not from a multi-cloud service base. It is challenging to efficiently find a composition solution in a multiple cloud base because it involves not only service composition but also combinatorial optimization. In this paper, we first propose a framework of service composition in multi-cloud base environments. Next, three different cloud combination methods are presented to select a cloud combination subject to not only finding feasible composition sequence, but also containing minimum clouds. Experimental results show that a proposed method based on artificial intelligence (AI) planning and combinatorial optimization can more effectively and efficiently find sub-optimal cloud combinations.

Keywords

Service Composition, Multi-Cloud Base, Cloud Combination, Composition Planner

1. INTRODUCTION

A web service is a modular, self-describing, self-contained and web-accessible software unit component that can be published by service provider and invoked by service requesters over the Internet. In recent years, people become more interested in web services because of its potential power in real applications. Thus an increasing amount of companies and organizations prefer to keep only their principle business, but outsource other application services over the Internet [1]. Nowadays, different kinds of web services have been published and made available for individual users and organizations. A huge amount of research works have been devoted to areas such as service discovery. The resulting techniques are currently powerful enough to help web service requesters find a standalone web service.

Nevertheless, in many cases to have a standalone web service is not enough. When there is no single service has the capability to satisfy a service requester's requirement, service composition is needed to select several correlative web services together for the purpose of fulfilling the service requester's goal. Therefore, the problem of how to effectively and efficiently compose existing web services has attracted a lot of research interests and is an important open problem.

In earlier research, service composition methods take an assumption that all selected web services found in the composition sequence come from the same service repository. In other words, these existing methods try to find composite web services just storing in one service repository, rather than those web services distributed in multiple different locations. However, with the emergence of some representative cloud computing platforms, such as Windows Azure Platform [3] and Amazon S3[4], it will be very common for service providers to publish their web services at different cloud platforms, which has distinctive advantages such as adaptivity, scalability and transparency of load scheduling. In such a case, when we need web services from multiple service repositories to obtain a valid composition, traditional methods may not find a solution.

In environments with multiple clouds involved, it is more likely that a service composer can find a sequence of web services than in an individual cloud. Thus in many scenarios, we need to find web services from multiple clouds, if a single cloud cannot give us all the services that we need. In such a situation, however, it is challenging for a service composer to find an appropriate composition sequence. The reason is that instead of a composition, we actually need to find a valid composition with the minimum number of clouds involved in, which is very important since the communication cost between two web services from different clouds would be expensive and time consuming. How to effectively and efficiently choose a service plan that minimizes the cloud used is an open issue.

In this paper, we address the problem of web service composition in environments with multiple clouds. To our best knowledge, this is the first work addressing this problem. Due to the expensive communication cost among web services from different clouds, the goal of our work is to effectively and efficiently minimize the number of clouds involved in a service composition sequence. It is a challenging problem because it couples planning according to

Annual International Conference on Cloud Computing and Virtualization (CCV 2010).
Edited by Prof. Gagan Agrawal.

ISBN: 978-981-08-5864-3.
doi:10.5176/978-981-08-5837-7_166

service providing constraints and optimization for cloud selection. The key idea of our work is to convert this problem to a set covering model, and find a sub-optimal cloud combination solution using an approximation algorithm, while using AI planning to compose web services. More specifically, we model a multiple cloud service base as a tree, use an approximation algorithm to optimize cloud selection, while employing an AI planning system for service composition. Our experimental results show that the proposed smart cloud combination method can effectively and efficiently find a desirable cloud combination for web service composition.

This paper is organized as follows. In Section 2, we review previous work. In Section 3, we present the problem formulation. In Section 4, we first present the overall framework of our semantic service composer in multiple cloud base environments, and then three different algorithms for web service composition in a multi-cloud environment. We give experimental results and analysis in Section 5 and conclude the paper in Section 6.

2. PREVIOUS WORK

In this section, we discuss some related works about the methods of web service compostion. We also look into the latest progress of cloud computing researches.

In terms of applied technology and theory foundation [1, 2], service composition methods can be divided into workflow-based composition, AI planning composition, and composition via program synthesis. Here, we focus on methods of web service composition by AI planning. It compiles a web service composition problem into an AI planning problem, where services are considered as actions in planning formulations.

There are several good composition planners. OWLS-XPlan [5, 6] is an OWL-S [7] service composition planner that has been applied to an agent based mobile eHealth system for emergency medical assistance (EMA) planning tasks. It first converts a service composition request, along with all candidate services described by OWL-S 1.1, into a composition problem plus a composition domain. It then executes a composition planner XPlan to find a composition plan. SHOP2 [8] is a Hierarchical Task Network (HTN) planning system, which can be used to do automatic composition of web services. All candidate services, described in OWL-S, are firstly translated into a SHOP2 domain by a sound and complete conversion algorithm. Then the SHOP2 planner recursively divides composition task into many subtasks until every subtask can be executed by a single web service. At last a composition plan can be generated to satisfy requester's composition goal. Hoffmann [9] introduced a planning formalism to represent web service composition, and also identified a special case of web service composition called "forward effects", where semantics become easier to deal with. An initial experiment has been conducted by using the conformant-FF planner [10]. Another service composer in [11] converts DAML-S [12] described services into Verb-Subject-Object (VSO) triples, and then constructs a sequence of atomic services to satisfy a user's service composition request.

Nowadays, cloud computing is becoming a prominent platform for providing web services. In the Industry, Microsoft has published its cloud computing platform, Windows Azure [3], which provides not only computing resources, but also cloud data storage centers. Other companies, like Amazon [4] and Salesforce [13], provide similar cloud computing infrastructures. In the academia, Zhang [14] proposed a cloud computing open architecture CCOA, and pointed out that virtualization and Service-Oriented Architecture (SOA) are the two key enabling techniques. Wang [15] introduced several enabling techniques for cloud computing.

In this paper, we tackle the problem of cloud selection for web service composition in a multiple cloud base, which is composed of different clouds. It exploits OWLS-XPlan [5] as service composer to generate a composition plan after a cloud combination is chosen by our methods.

3. PROBLEM FORMULATION

In this section, we first define what web service and web service composition are. Next, we present the formulation of web service composition in the cloud computing setting, and the corresponding metrics that we will optimize for the purpose of boosting the efficiency of service composition.

Definition 1 (Web Service). *A web service is a 2-tuple* $\langle I, O \rangle$, *where both I and O are service interfaces. An interface is a set of propositions. Given an interface J, service* $\langle I, O \rangle$ *is applicable to J if and only if* $I \subseteq J$. *The resulting interface, after* $\langle I, O \rangle$ *applied, is* $J \setminus I \bigcup O$.

Note that an interface could be of complicated expressions, not only each proposition has a data type, but also a name. Here we assume an interface consists of ontology concepts, although in practice we handle much more complicated service interfaces.

Definition 2 (WSC Problem). *A* ***web service composition (WSC) problem****, in a general environment setting, is defined as* $\langle I, G, S \rangle$, *where*

1. I *is an initial interface, provided by a user in its request, indicating the starting point.*
2. G *is a goal interface, provided by a user in its request, indicating the ultimate interface the user wants to obtain.*
3. S *is a set of candidate web services.*

Given a web service composition problem $\langle I, G, S \rangle$, a solution to this problem, called a composition plan π, is a sequence of totally ordered web services such that $\pi \subseteq S$. By applying each service in π, the resulting interface is a superset of G.

Definition 3 (Service Provider). *A service provider publishes a set of web services, which corresponds to a service file sf* where $sf = \{s_1, s_2, ..., s_n\}$, and $\forall s_i (1 \le i \le n)$ *is a web service from the service provider.*

In a real world application, service provider often specifies their published web services in a standard OWL-S specification file.

Definition 4 (Cloud Service Base). *A cloud service base C is a set of service files where* $C = \{sf_1, sf_2, ..., sf_m\}$, $\forall sf_i (1 \le i \le m)$ *is a service file published by a service provider.*

A cloud service base usually refers to many service files. Nowadays, there are many clouds on the market. They give us the underlying infrastructure for fulfilling a user's service requests.

Definition 5 (Multiple Cloud Base (MCB)). *A multiple cloud base is a set of clouds, such that* $MCB = \{C_1, C_2, ..., C_N\}$, *where each* $C_i (1 \le i \le N)$ *is an independent cloud service base.*

In a MCB environment, web services in different commercial cloud computing platforms, can be used together via mutual communication to satisfy a complex service request.

Definition 6 (Cloud WSC Problem). *A* ***cloud WSC problem*** *is defined as* $\langle I, G, MCB \rangle$, *where*

1. *I and G are the same as those in the general web service composition.*
2. *MCB is a multiple cloud base.*

A composition for a cloud WSC problem is a sequence of <*service*, *cloud*> pairs, $W = \{< s_i, C_p >, < s_j, C_q >, ..., < s_k, C_r >\}$, such that applying the sequence of services results in an interface R, $G \subseteq R$.

An optimal composition to a cloud WSC problem is a sequence of <*service*, *cloud*> pairs, $W' = \{< s_i', C_p' >, < s_j', C_q' >, ..., < s_k', C_r' >\}$, with the minimum $|\{C' | \exists s', s' \in W' \wedge s' \in C'\}|$. In other words, an optimal service composition has the minimum number of clouds involved in service composition sequence.

In order to make the problem more explicit, we give a specific example. Suppose that there is a MCB consisting of four clouds $\{C_1, C_2, C_3, C_4\}$. Each of these has a set of service files, which is a subset of $\{a, b, c, d, e\}$. Moreover, each service file contains a certain number of web services. Thus, a service file is mapped into a service provider who publishes one kind of web services. In particular, $\{a, b, c, d, e\}$ respectively corresponds to {"EMA services", "Medical Flight Company Services", "Medical Transport Company Services", "Non Medical Flight Company Services", "Non Medical Transport Company Services"} in the emergency medical assistance (EMA) planning tasks, called Health-SCALLOPS, which is also used as experiment data in [5, 6]. Table 1 shows this multiple cloud base and the distribution of its cloud service base.

Table 1. A multiple cloud base and its cloud distribution

Cloud	C_1			C_2		C_3		C_4			
Service file	a	b	c	d	e	c	d	a	b	c	e
Services number	2	3	8	3	3	8	3	2	3	8	3

In the above MCB environment, there exist different cloud combination situations when a service requester provides initial and goal description of service composition requirement. For example, if a service file set {a, b, e} is required to be included in a requester's service composition request, then $\{C_1, C_2, C_3, C_4\}$, $\{C_1, C_2, C_3\}$, $\{C_1, C_2, C_4\}$, $\{C_1, C_3, C_4\}$, $\{C_2, C_3, C_4\}$, $\{C_1, C_2\}$, $\{C_1, C_4\}$, $\{C_2, C_4\}$, $\{C_3, C_4\}$ and $\{C_4\}$ are all the candidates of a valid cloud combination.

4 CLOUD COMBINATION ALGORITHMS

4.1 Service Composer Overview

In a multiple cloud base environment, the service composer consists of several correlative modules, including cloud combiner, composition convertor, composition planner, service ontologies and a MCB environment. Its overall structure is illustrated in Figure 1.

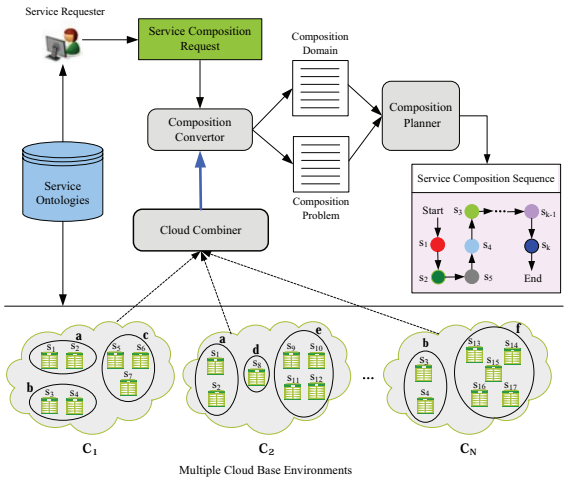

Figure 1. Service composer in MCB environments.

Service requester first provides initial and goal descriptions of service composition request using service ontologies. Next, the cloud combiner selects appropriate cloud combination from the MCB using a combinatorial optimization method. After that, the composition convertor transforms the composition request and the selected cloud combination into a composition domain and a composition problem. Finally, a composition planner executes and finds a composition plan, which is composed of a service composition sequence that can satisfy the requester's service composition goal.

When generating a service composition plan, the cloud combiner tries to pick the most appropriate clouds for service conversion and planning. The quality of the selected cloud set is important, because it affects not only the number of clouds involved in the final service composition sequence, but also the time efficiency to obtain a good composition plan.

We propose three different methods for selecting the cloud combination: (1) all clouds combination method, (2) base cloud combination method, and (3) smart cloud combination method.

4.2 All Clouds Combination Method

In this combination method, we put all clouds in *MCB* as inputs for composition conversion. The description of this process is shown in Algorithm 1.

In this algorithm, it first coverts all web services involved in *MCB* and composition request into a composition domain and a composition problem, respectively. Next, it executes a composition planner to find a planning solution. If there is a composition plan sequence, it is returned to the service requester.

Algorithm 1: All Clouds Combination

Input: *MCB*, composition request *compReq*;
Output: composition sequence *compSeq* or NULL;

```
1. cur_combSet ← MCB ;  /*get cloud combination set
2. (cur_combSet, compReq) → (Domain, Problem) ;
3. compSeq ← NULL ;
4. Exec composition planner;
5. get compSeq;
6. if (compSeq is not NULL) then {
7.    return compSeq;
8. }
9. return NULL;
```

We take a MCB in Table 1 as an example. For a composition request, suppose that $\{a, b, c, d\}$ is needed to satisfy its functionality requirement. Using OWLS-XPlan as the composition planner, the *all clouds combination* method can find a composition sequence $\{< s_i, C_j > | s_i \text{ is a service}, C_j \in MCB\}$. In such a case, it contains twelve web services and three different clouds $\{C_1, C_3, C_4\}$ in the found service composition sequence.

Time Complexity. We assume it takes $O(1)$ time to execute the composition planner one time, and convert each web service as well. The time complexity of this method is determined by its composition conversion for all web services involved in the MCB. For a multiple cloud base $MCB = \{C_1, C_2, ..., C_N\}$, we assume that each cloud $C_i\ (1 \le i \le N)$ has L_i service files and M_i web services. Let $L = Max\{L_1, L_2, ..., L_N\}$, $M = Max\{M_1, M_2, ..., M_N\}$. In order to calculate its time complexity, we need to calculate the total number of web services involve in MCB: $\sum_{i=1}^{N} M_i$. So the whole time complexity of the *all clouds combination* method is $T(MCB) = O(\sum_{i=1}^{N} M_i) = O(N * M)$.

4.3 Base Cloud Combination Method

Although the *all clouds combination* method can find a service composition sequence quickly, it does not minimize the number of clouds in the final service composition sequence. It can result in a serious issue because web services distributed in different clouds remarkably increase communication cost and financial charges.

We give another cloud combination method, called the base cloud combination method, which can find an optimal cloud combination with a minimum number of involved clouds. It is described in Algorithm 2. In this algorithm, it recursively enumerates all different cloud combination possibilities until a composition solution is found in a cloud combination. More specifically, it starts from testing all singleton sets of clouds and stops if a valid composition plan can be found using a single cloud, Otherwise, it tests cloud sets of size two, three, etc., until reaching a cloud set from which a service plan can be found. In Algorithm 2, ***RecurComb*** is a recursive algorithm designed to find all cloud combinations containing k number of clouds from a total number of S clouds.

Algorithm 2 Base Cloud Combination

Input: *MCB*, clouds number *N*, composition request *compReq*;
Output: composition sequence *compSeq* or NULL;

```
1.  foreach i = 1 to i ≤ N do {
2.     combList ← NULL ;  /*initialize cloud combination list
3.     RecurComb( MCB, N, i, NULL, 1 );
4.     get C_N^i cloud combinations in combList;
5.     foreach j=1 to j ≤ C_N^i do {
6.        cur_combSet ← combList(j) ; /*get a combination
7.        (cur_combSet, compReq) → (Domain, Problem) ;
8.        compSeq ← NULL ;
9.        Exec composition planner;
10.       get compSeq;
11.       if (compSeq is not NULL) then
12.          return compSeq;
13.    }
14. }
15. return NULL;
```

RecurComb($MCB, S, k, cloudSet, pos$)

```
1. if ( k == 1 ) then {
2.    foreach m = pos to m ≤ pos + S − 1 do
3.       add (cloudSet ∪ MCB[m]) into combList ;
4.    return;
5. }
6. foreach l = 1 to l ≤ S − k + 1 do {
7.    tempCloudSet ← (cloudSet ∪ MCB[pos + l - 1]) ;
8.    RecurComb( MCB, S - l, k-1, tempCloudSet, pos+l );
9. }
```

Taking the same $MCB = \{C_1, C_2, C_3, C_4\}$ in Table 1 as an example, suppose that $\{a, b, c, d\}$ is needed to satisfy a service requester's functionality requirement, the *base cloud combination* method first tries all single cloud combinations. In other words, it checks C_1, C_2, C_3 and C_4. However, none of them can satisfy the service request because of their insufficient service files to satisfy $\{a, b, c, d\}$. Then it begins to examine cloud combinations with two clouds. The cloud combination $\{C_1, C_2\}$ is selected to generate a composition sequence for the service requester, because it includes all service files $\{a, b, c, d\}$ needed by the service requester.

Time Complexity. Considering the worst case behavior, it will check all cloud combinations in order to find a composition sequence. The time complexity is also determined by its composition conversion for all involved web services. Under the same notation as in our previous analysis, for each kind of cloud combinations with $i\ (1 \le i \le N)$ clouds, there are C_N^i different cloud combinations, each of which has $(M_{j1}+M_{j2}+...+M_{ji})$ web services, where $1 \le j \le C_N^i$. So, time complexity of the *base cloud combination* method is: $T(MCB) = O(\sum_{i=1}^{N} \sum_{j=1}^{C_N^i} (M_{j1}+M_{j2}+...+M_{ji}))$

$= O(\sum_{i=1}^{N}\sum_{j=1}^{C_N^i}(i*M)) = O(\sum_{i=1}^{N}(i*M*C_N^i)) = O(N*M*\sum_{i=1}^{N}C_N^i)$

$= O(N*M*2^N)$.

4.4 Smart Cloud Combination Method

Although the *base cloud combination* method can generate the optimal cloud combination, which contains the minimum number of clouds, its time complexity is very high (exponential in the number of clouds) since it needs to enumerate all possible cloud combinations in the worst case. In order to quickly find a good combination plan, we present a s*mart cloud combination* method. It is designed for efficiently finding a near optimal cloud combination based on an approximative algorithm.

The *smart cloud combination* method first models a *MCB* as a tree, and then finds a minimum request set by searching in the *MCB* tree.

4.4.1 Modeling a MCB as a tree

Considering the whole *MCB* as the root node, a MCB can be modeled as a tree. Taking the MCB in Table 1 as an example, part of its tree is shown in Figure 2. Due to limited space, some of the service file nodes connected with dashed lines, in the third level, do not show their corresponding web services in the last level of the tree.

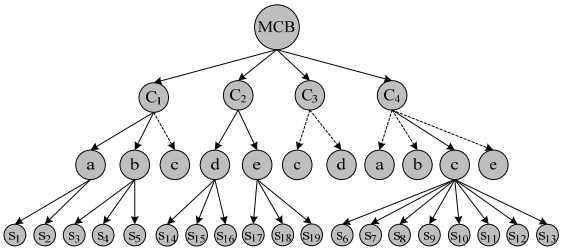

Figure 2. Part of the tree for a multiple cloud base.

There are four levels in a *MCB* tree. The first level represents a complete MCB. The second level corresponds to individual cloud service bases, e.g., it has C_1, C_2, C_3, C_4 in the above *MCB* tree. The third level models service files, e.g. a, b, c, d, e are included in the above *MCB* tree. The last level models web services, e.g., it contains web services s_1, s_2, s_2, ..., s_{19} in the above *MCB* tree. In many cases, a service provider may publish his service file in many different clouds, e.g., service file a is published to C_1 and C_4.

4.4.2 Finding the minimum request set

Definition 7 (Minimum Request Set (MRS)). *For a MCB , given a composition request* $\langle I, G\rangle$ *that maps to a service sequence* $Seq = \{s_i, s_j, ..., s_k\}$ *, which is part of a composition sequence* $W = \{< s_i, C_p>, < s_j, C_q>, ..., < s_k, C_r>\}$ *. A* MRS *is a service file set, where*

1. $Min\left|\left\{sf \,\middle|\, Seq \subseteq \bigcup_{sf \in MRS} GetChilds(sf)\right\}\right|$.
2. *For* $\forall sf \in MCB$, $GetChilds(sf)$ *is the function to expand all its web services in a* MCB *tree.*

MRS contains the indispensable service files for a service request, but not other service files. For example, suppose that a service requester provides a composition request $\langle I, G\rangle$, and it needs a service sequence $Seq = \{s_1, s_2, s_4, s_7, s_9, s_{14}, s_{16}\}$ as shown in Figure 2. Its minimum request set is $MRS = \{a, b, c, d\}$.

Given a composition request, its *MRS* can be found by searching in a *MCB* tree, as described in Algorithm 3.

Algorithm 3 Finding a Minimum Request Set

Input: *MCB*, composition request *compReq*;
Output: Minimum Request Set *MRS* or NULL;

```
1.  MRS ← NULL ; /*initiate MRS
2.  compSeq ← NULL ; /*initiate composition sequence
3.  Invoke All Clouds Combination;
4.  get compSeq;
5.  if (compSeq is not NULL) then {
6.      foreach < si, Cj > ∈ compSeq do {
7.          search Cj in MCB tree;
8.          find sf ∈ Cj ;
9.          if ( sf is not in MRS ) then
10.             MRS ← MRS ∪ {sf} ;
11.     }
12.     return MRS ;
13. }
14. return NULL;
```

The all clouds combination method is first invoked to generate a composition sequence $\{< s_i, C_j > | s_i \text{ is a service}, C_j \in MCB\}$. Next, to reduce the tree search space, a subtree with root node C_j is located based on each pair $< s_i, C_j >$, after that a service file sf $(s_i \in sf)$ can be quickly found out within the subtree C_j . Finally, all searched service files constitute the minimum request set *MRS*.

For the above example, suppose that, after invoking *all clouds combination*, the composition planner finds $compSeq = \{<s_1, C_1>, <s_2, C_1>, <s_4, C_4>, <s_7, C_3>, <s_9, C_3>, <s_{14}, C_2>, <s_{16}, C_2>\}$. When dealing with $<s_1, C_1>$, the search space is reduced to the subtree C_1, and the service file a can be identified according to service s_1. Other service files b, c and d are found and added into *MRS* in the same way. Finally, $\{a, b, c, d\}$ is determined as *MRS*.

Time Complexity. The time complexity for finding MRS consists of two parts. The first part is due to the *all clouds combination* method, which is $O(N * M)$. The second part is to find service files in the *MCB* tree. Suppose that it contains t services in a composition sequence. Considering the worst case behavior, for each $< s_i, C_j >$, it needs to search the whole space of subtree C_j to find its service file sf . It takes $O(N)$ time to find subtree C_j and $O(M)$ time to find sf . Comparing to M , the composite services

number t is smaller. So the time complexity of Algorithm 3 is $T(MCB) = O(N * M + t * (N + M)) = O(N * M)$.

4.4.3 Objective functions and constraint

Given a *MRS*, a cloud is more likely to be selected as a candidate in the optimal cloud combination, if all of its service files are included in *MRS*. On the other hand, if a cloud includes extra service files that are not in the *MRS*, these files are not useful but cause additional computational cost for the service composition convertor and composition planner.

Definition 8 (Cloud Cost). $MCB = \{C_1, C_2, ..., C_N\}$. *Given a* MRS, *each cloud* $C_i\ (1 \le i \le N)$ *has a cloud cost* $SN(C_i)$ defined as:

$$SN(C_i) = \begin{cases} \sum_{sf \in (C_i - MRS)} |sf| & , \ if\ (C_i\text{-}MRS) \neq \varnothing \\ 0 & , \ otherwise \end{cases}$$

For a cloud C_i, its cost is calculated by the total number of web services in service files, which are not included in MRS. For example, in Figure 2, given a $MRS=\{a, b, c, d\}$, the costs of clouds are: $SN(C_1) = 0$, $SN(C_2) = |e| = 3$, $SN(C_3) = 0$, $SN(C_4) = |e| = 3$.

Definition 9 (Smart Cloud Combination Problem). *Given a MRS and* $MCB = \{C_1, C_2, ..., C_N\}$, *a smart cloud combination problem is the following combinatorial optimization problem with binary variables* $Sel(C_i)$, $i = 1, ..., N$:

1. *Objective functions*

$$\begin{cases} (1)\ \ Min \sum_{i=1}^{N} Sel(C_i) \\ (2)\ \ Min \sum_{i=1}^{N} (SN(C_i) | Sel(C_i) = 1) \end{cases}, \quad Sel(C_i) = \begin{cases} 1, & if\ C_i\ is\ selected \\ 0, & otherwise \end{cases}$$

2. *Constraint condition*

$$MRS \subseteq \bigcup_{i=1}^{N} \{sf \,|\, sf \in C_i, Sel(C_i) = 1\}$$

A cloud combination is composed of the selected clouds ($Sel(C) = 1$). The first goal is to minimize the number of clouds involved in the cloud combination, after that we try to optimize the total cost of selected clouds ($SN(C)$). The constraint condition is that all service files in MRS must be included in the selected clouds.

4.4.4 The smart cloud combination algorithm

Based on finding *MRS* and formalizing it as a combinatorial optimization problem, smart cloud combination is described in the following algorithm 4.

It takes four steps in this algorithm, step one is to find *MRS*, which has been discussed. In particular, there does not exist a cloud combination satisfying user's request, if its *MRS* cannot be found in this step. In step two, we preprocess each cloud in *MCB* to reduce its service files space, and set its cloud cost. More specifically, for each service file sf involved in a cloud C, if it is also included in *MRS*, then sf is added into the reduced cloud C'. Otherwise, sf is removed from the reduced cloud C', and the number of services involved in sf is added to the cost of reduced cloud C'. After that, all clouds are transformed to their corresponding reduced clouds, which are stored in a cloud reduced set *cloudReducSet*. At the same time, each reduced cloud C' has a cloud cost $SN(C')$.

Algorithm 4 Smart Cloud Combination

Input: *MCB*, composition request *compReq*;
Output: composition sequence *compSeq* or NULL;

```
/*Step one. finding MRS*/
1.  Invoke Finding Minimum Request Set;
2.  if ( MRS is NULL) then
3.    return NULL;
/*Step two. cloud reduction and cloud cost setting*/
4.  cloudReducSet ← ∅ ;
5.  foreach Ci ∈ MCB do {
6.    Ci' ← ∅ ; SN(Ci') = 0 ;
7.    foreach sf ∈ Ci do {
8.      if ( sf is in MRS ) then
9.        Ci' ← Ci' ∪ {sf} ;
10.     else if ( sf is not in MRS ) then
11.       SN(Ci') = SN(Ci') + |sf| ;
12.     end
13.   }
14.   cloudReducSet ← cloudReducSet ∪ Ci' ;
15. }
/*Step three. finding cloud combination*/
16. combSet ← ∅ ;
17. while ( cloudReducSet is not ∅ ) do {
18.   maxNum ← 0 ; minCost = 0 ; selFlag = 0 ;
19.   foreach Ci' ∈ cloudReducSet do {
20.     if ( maxNum < |Ci'| ) then
21.       maxNum ← |Ci'| ; minCost = SN(Ci') ; selFlag = i ;
22.     else if ( (maxNum == |Ci'|) ∧ (SN(Ci') < minCost) ) then
23.       minCost = SN(Ci') ; selFlag = i ;
24.     end
25.   }
26.   combSet ← combSet ∪ MCB[selFlag] ;
27.   selCloud ← cloudReducSet[selFlag] ;
28.   foreach Ci' ∈ cloudReducSet do
29.     cloudReducSet[i] ← cloudReducSet[i] − selCloud ;
30. }
/*Step four. getting composition sequence*/
31. (combSet, compReq) → (Domain, Problem) ;
32. Exec composition planner;
33. return compSeq;
```

Based on approximate algorithms [16, 17, 18] for solving set covering problems, the third step is to find a sub-optimal cloud combination by using the cloud reduced set and reduced cloud costs. We implement an approximate algorithm based on [17], and also expand it by considering the reduced cloud cost into the

process of cloud selection. We check each reduced cloud C' in *cloudReducSet* , if the number of its included service files is larger than the current maximum reduced cloud, then C' is selected as the current maximum reduced cloud. Meanwhile, if the number of service files in C' is equal to the current maximum reduced cloud, we further examine whether its cloud cost is lower than that of the currently selected reduced cloud. If that is true, we replace the current maximum reduced cloud by C'. So we add the cloud C that is in *MCB* and corresponds to C' into the final cloud combination. After that, all reduced clouds in the reduced cloud set are updated by deleting their service files, which also exist in selected C'. This process stops when all reduced clouds have been updated to $\varnothing$. At the end of this step, a cloud combination is found with approximately minimum number of clouds. In the last step, all service files involved in the cloud combination, along with composition request, are converted into a planning domain and problem, which is solved by an AI planner for service composition. We know a solution composition plan can be found since the *MRS* is covered by the selected clouds.

Taking *MCB* in Figure 2 as an example, its clouds are distributed as: $C_1 = \{a, b, c\}$, $C_2 = \{d, e\}$, $C_3 = \{c, d\}$, $C_4 = \{a, b, c, e\}$. For a requester's composition request, in step one, we suppose that it can find $MRS = \{a, b, c, d\}$. In step two, it converts all these clouds to their corresponding reduced clouds and costs. According to the found MRS, $C_1' = \{a, b, c\}$, $C_2' = \{d\}$, $C_3' = \{c, d\}$, $C_4' = \{a, b, c\}$. $SN(C_1') = 0$, $SN(C_2') = |e| = 3$, $SN(C_3') = 0$, $SN(C_4') = |e| = 3$. In step three, for the first time loop, although C_4' has also three service files, its cost is larger than C_1'. So C_1' is selected as the maximum reduced cloud, and C_1 is added into cloud combination. At the end of this loop, all reduced clouds are recalculated by subtracting C_1'. Thus, we can get $C_1' = \varnothing$, $C_2' = \{d\}$, $C_3' = \{d\}$, $C_4' = \varnothing$. For the second time loop, C_2' and C_3' both have one service file. However, C_3' has a lower cost than C_2'. So C_3' is selected as the maximum reduced cloud, and C_3 is added to the cloud combination. Then all reduced clouds are recalculated again by subtracting C_3'. At this time, all reduced clouds are $\varnothing$. Finally, we get the cloud combination $\{C_1, C_3\}$ and feed them as inputs to an AI planner to get a composition sequence.

Time Complexity. Considering the worst case behavior of the smart cloud composition method, its time complexity consists of four parts. The first part is due to *Finding Minimum Request Set*, which is $O(N * M)$. The second part is the time used to reduce the cloud dimension, which is $O(\sum_{i=1}^{N} L_i) = O(N * L)$. The third part is to generate cloud combination. It takes $O(N)$ time to select the maximum number of cloud for each time. The worst case is to select all N clouds as the cloud combination, so it takes $O(N^2)$ time to generate the cloud combination. In the last step, the found cloud combination has $|combSet|$ number of clouds, so it takes $O(|combSet| * M)$ time to convert all selected services and find a service composition sequence. Comparing to M and N, L and $|combSet|$ are smaller. Hence, the total time complexity is $T(MCB) = O(N * M + N * L + N^2 + |combSet| * M) = O(N * M + N^2)$. We see that its time complexity is quadratic, much better than the exponential complexity of the base method. On the other hand, the quality of the smart method is nearly optimal, much better than the *all clouds combination* method. Hence, the smart method achieves a superior tradeoff of solution time and quality, and is a practical solution for deployment in multi-cloud web service provision environments.

5 EVALUATION

5.1 Experiment Environment and Data

In order to evaluate the effectiveness and efficiency among our proposed three cloud combination methods for web service composition, we have developed a prototype system, called Multi-Cloud Service Composer, based on OWLS-XPlan [5]. All three cloud combination methods are implemented to help requesters select clouds in MCB environments. Experiments are conducted on a DELL Server with Pentium IV 2.4GHZ CPU and 1G RAM. We have used Eclipse 3.2 as the prototype development platform, and Java as the programming language to implement the cloud combiner module in Figure 1.

Our experimental data comes from default web services test set in the OWL-S XPlan package. There are in total five service files, each of which has several web services. In order to simplify them, we mark $\{a, b, c, d, e\}$ as each of the original service files. Moreover, {2, 3, 8, 3, 3} represents the number of web services involved in each of the service files, respectively.

We simulate a dynamic MCB environment, where there are four clouds $MCB = \{C_1, C_2, C_3, C_4\}$. Each cloud has some service files from different service providers. Here, we test five different MCB configurations as the test data set, as shown in Table 2.

Table 2 Five MCB settings

MCB	C_1	C_2	C_3	C_4
MCB 1	{a, b, c}	{d, e}	{c, d}	{a, b, c, e}
MCB 2	{a, b}	{c}	{b, e}	{a, d, e}
MCB 3	{a, c, e}	{e}	{a, b}	{c, d}
MCB 4	{b, c, e}	{c, d}	{a, b, c}	{d, e}
MCB 5	{a, b}	{b, c}	{c}	{a, d, e}

5.2 Experimental Results and Analysis

In order to evaluate the three different cloud combination methods, a service composition request is given to the cloud combiner. Suppose that a service file set {a, b, c, d} is needed to satisfy the service composition request <*I*, *G*>.

For the {a, b, c, d}, each cloud combination method selects clouds and generates a cloud combination from each MCB. The experimental results are shown in Table 3, which gives the cloud combination (*Comb*) and the converted (*Conv*) services number.

Table 3 Comparisons among three combination methods

MCB	*All clouds*		*Base cloud*		*Smart cloud*	
	Comb	*Conv*	*Comb*	*Conv*	*Comb*	*Conv*

MCB 1	$C_1 C_3 C_4$	46	$C_1 C_2$	65	$C_1 C_3$	70
MCB 2	$C_1 C_2 C_3 C_4$	27	$C_1 C_2 C_4$	148	$C_1 C_2 C_4$	48
MCB 3	$C_1 C_3 C_4$	32	$C_3 C_4$	128	$C_3 C_4$	48
MCB 4	$C_1 C_2 C_3 C_4$	44	$C_2 C_3$	68	$C_2 C_3$	140
MCB 5	$C_1 C_2 C_3 C_4$	32	$C_2 C_4$	112	$C_1 C_2 C_4$	56

We have following observations from Table 3.

(1) The *all clouds combination* method can quickly find a composition sequence. However, it always finds a cloud combination with the largest number of clouds..

(2) The *base cloud combination* method can find the optimal cloud combination containing the minimum number of clouds. However, it will result in serious issue when the number of *MCB* becomes larger and larger, because its time complexity $O(N * M * 2^N)$ increases exponentially.

(3) The *smart cloud combination* method finds a cloud combination by using an approximative algorithm. In all but one case, it can find an optimal cloud combination. From the Table 3, we can see that, this method is nearly as good as the second one, although its time complexity is much lower.

(4) Since the number of conversions (*Conv*) directly relates to execution time, we can see from Table 3 that the base method requires a much larger conversion number than the smart method. Hence, the smart method is much more efficient.

6. CONCLUSIONS

In this paper, we define and study the problem of web service composition in a multiple cloud base (MCB) environment. As cloud based services become increasingly popular, service composition algorithms that are aware of the cloud selection have deep impacts to the overall efficiency improvement and cost saving for enterprises. We first propose a framework of web service composition in MCB environments. Then, three different cloud combination methods are represented to help service requesters select cloud combination. Experimental results indicate that our proposed method based on artificial intelligence (AI) planning and combinatorial optimization can more efficiently and effectively find high quality service composition plans with minimum cloud expense. The proposed method achieves a superior tradeoff of solution time and quality, and is a practical solution for deployment in multi-cloud web service provision environments.

7. ACKNOWLEDGMENTS

We would like to greatly thank Matthias Klusch and Andreas Gerber for their providing the open source of OWLS-XPlan, and the anonymous reviewers for giving insightful comments on this paper. This work is supported by NSF grant IIS-0713109 and a Microsoft Research New Faculty Fellowship.

8. REFERENCES

[1] Rao, J. and Su, X. 2005. A survey of automated web service composition methods. Lect. Notes. Comp. Sc. 3387, 43-54.

[2] Srivastava, B. and Koehler, J. 2003. Web service composition-current solutions and open problems. In Proceedings of the ICAPS workshop on planning for web services.

[3] Windows Azure Platform. Microsoft cloud computing platform: http://www.microsoft.com/windowsazure/

[4] Amazon S3. Amazon Simple Storage Service cloud computing platform: http://aws.amazon.com/s3/

[5] Klusch, M. and Gerber, A. 2006. Fast composition planning of OWL-S services and application. In Proceedings of the 4th Eruopean Conference on Web Services. 181-190.

[6] Klusch, M. and Gerber, A. and Schmidt, M. 2005. Semantic web service composition planning with OWLS-XPlan. In Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web.

[7] The OWL Services Coalition. 2004. OWL-S: semantic markup for web services. http://www.daml.org/services/owl-s/1.0/owl-s.html

[8] Sirin, E., Parsia, B., Wu, D., et al. 2004. HTN planning for web service composition using SHOP2. J. Web. Semant. 1, 4, 377-396

[9] Hoffmann, J., Bertoli, P., Pistore, M. 2007. Web service composition as planning, revisited: In between background theories and initial state uncertainty. In Proceedings of the AAAI. 22, 2, 1013-1018.

[10] Hoffmann, J. and Brafman, R. I. 2006. Conformant planning via heuristic forward search: A new approach. Artif. Intell. 170, (6-7), 507-541.

[11] Sheshagiri, M., DesJardins, M. and Finin, T. 2003. A planner for composing services described in DAML-S. In Proceedings of International Conference on ICAPS Workshop on planning for web services.

[12] Ankolekar, A., Burstein, M., Hobbs, J., et al. 2002. DAML-S: web service description for the semantic web. In Proceedings of the 1st International Semantic Web Conference. 348-363.

[13] Force.com. Salesforce cloud computing platform: http://www.salesforce.com/platform/

[14] Zhang, L. J. and Zhou, Q. 2009. CCOA: Cloud computing open architecture. In Proceedings of the 2009 IEEE International Conference on Web Services. 607-616.

[15] Wang, L., Tao, J., Kunze, et al. 2008. Scientific cloud computing: Early definition and experience. In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications.

[16] Mannino, C. and Sassano, A. 1995. Solving hard set covering problems.Oper. Res. Lett. 18, 1, 1-5.

[17] Johnson, D.S. 1973. Approximation algorithms for combinatorial problems. In Proceedings of the 5th annual ACM Symposium on Theory of computing. 38-49.

[18] Caprara, A. and Fischetti, M. and Toth, P. 1999. A heuristic method for the set covering problem. Oper. Res. 47, 5, 730-743.