CLUSTERING-BASED SERVICE SELECTION FOR DYNAMIC SERVICE COMPOSITION

Aram AlSedrani and AmeurTouir

College of Computer and Information Sciences, Department of Computer Science King Saud University, Saudi Arabia

ABSTRACT

The increase in the number of available web services led to the increase in the similarity of services functionality offered by different providers each with different QoS parameters. Therefore, in web service composition, the selection of the optimal service to satisfy the QoS values required by user is one of the significant requirements. Moreover, the dynamic nature of web services adds more challenges to obtain the accuracy of the selection process. Most of the existing service composition approaches deal with services changes during composition execution, causing a re-planning or re-selection that affecst the service composition performance. In this paper, we introduce the clustering-based service selection model that outperforms the existing ones. The proposed model has the ability to detect and recover the changes in service repository by monitoring the composition process from a global point of view. The approach is a two-levels-based web service clustering. The proposed model encompasses a clustering process, a planning process, a selection process and a recovery process.

KEYWORDS

Web services, service clustering, service composition, service selection, self-healing composition

1. INTRODUCTION

Web service composition involves the integration of several existing web services to provide more complex and powerful service. The goal of service composition is to reuse existing web services and composing them into a process. However, the process of composition is considered as a high complex task due to many reasons. According to (Rao & Su 2004) the complexity of service composition raises because of three main reasons. First, the huge increase on the number of web services available over the internet, whichcauses the increase of the service repositories available for searching. Moreover, web services is in continuously changing and updating which requires a dynamic composition at runtime causing intensive computations. Another reason when different providers develop web services in different models making the mapping between services in the compositions a difficult task.

Service selection is about choosing the most appropriate service for each task in the composition in order to satisfy user requirement.The input of service selection problem is the set of candidate services for various tasks involved in the composition plan. A single candidate services set consists of services providing the same functionality offered by different providers through service Quality of Service (QoS) profile. Moreover, the same provider may offer several services that have the same functionality with different QoS to obtain satisfaction of a large panoply of users(Moghaddam & Davis 2014).

Nowadays, the composition environment becomes more dynamic due to the increase in the number of web services that are frequently changing. Therefore, the need for self-adapted

DOI : 10.5121/ijwest.2017.8201

composition methods that acts according to environment changes is advocated. Some existing composition approaches consider more or less the dynamic environment context while executing services. This is performed by monitoring the changes that occur to services, and considering the environment as static in early phases as in (Friedrich et al. 2010) and (Wiesner et al. 2009).

A limited number of researches highlight the importance of monitoring the changes on services during the selection phase. Batakat(Barakat et al. 2012) proposes an algorithm that reacts to changes that occurred in the participating web services during the selection phase. The algorithm modeled the problem as a graph, where each node stores the optimal path from the started node. When a change occurs, the algorithm re-tracks the stored information in each node in the path and reselects a new service accordingly. The proposed algorithm runs in a dynamic environment. However, when the number of nodes increases the complexity and the storage capacity increases considerably.

The limited number of researches in this area indicates that the service selection in dynamic environment is a new research direction, and it is not mature enough to satisfy composition process requirements. The need for new techniques is highly recommended, especially with the fast growth of the web.

In this paper, we propose the self-healing model for web service selection that has the ability to recover from service failure. The self-healing model involves three main processes: the web service clustering process, the service selection process, and the failure recovery process. The service clustering process involves two levels of clustering: first, classify the similar web services based on their functionality to “task”, second, generate composite tasks and group them into more abstract representation called “job”. By two levels services clustering, we aim at three goals: First, minimizing the selection problem search space within jobs rather than services. Second, the ability of fast recovery, in case of service failure, by substitutes the faulty service from the corresponding job and task clusters. Third, isolate the faulty web service in its job boundary and not affecting the rest of the well composed service. The selection in the proposed model is monitored through its entire process, where the failure recovery process is triggered whenever an environment change is occurred.

2. THE SELF-HEALING MODEL FOR WEB SERVICE SELECTION:

We define the Web Service Selection Problem (WSSP) as finding the most suitable web services from a set of candidate web services that match the QoS attributes asked by user. The adopted self-healing model embodies the service clustering process, the service selection process, and the failure recovery process. The service clustering classifies the web services with similar precondition and post condition to tasks. Thus, each task includes a set of atomic web service. As a second level, the service clustering classifies similar tasks into more abstract representation named jobs. A job represents a set of simple or composite tasks that have the same precondition and post condition. A composite task is composed of a sequence of tasks. The selection process is responsible of selecting the best candidate web service (low level) related to each job (high level) involved in the planning phase based on its QoS values. However, if any contingency occur during the selection process, the failure recovery process is triggered to react and solve the emerging problems.

As it was stated in the formulation of the WSSP, we used two types of service clustering in order to arrange the services prior the selection process: the task clustering and the job clustering. Each (J), includes multiple candidate tasks either atomic or composite. The principle of web service selection algorithm is to substitute each job from the composition plan by a corresponding web service via the clustered tasks in order to achieve the requested service (RS) and satisfy the QoS

requirements. The input of the selection algorithm is the QoS Constraint Plan (QCP) that consists of set of connected jobs, and the solution is the Execution Plan (EP) that consists of set of connected web services. The QoS attributes that are going to be addressed by our model are the Cost (C) and the Execution Time (ET). The Cost (C) is the price that a user has to pay to the service provider for theinvocation of a service (ω). The Execution Time(ET)is the time measured from invoking (ω) until theresult is received(Rajeswari et al. 2014). Moreover, the failure recovery algorithm is triggered in case of web service failure. It aims at collaborating with the selection process to recover the failures by substituting the faulty web service with a valid service from the same job cluster. In the sequel, we will elaborate the self-healing model processes in more detail.

2.1. DEFINITION AND SPECIFICATION:

In the sequel, we use the following definitions and notations:

- A web service directory WS is the set of services of a specific domain that exist in a service repository. We assume that each web service ω∈ WS is atomic that is not composed of other web services. Each ω∈ WS has two values: a cost Cthat specifies the cost of executing ω, the execution time ET that denotes the execution time ofω.

- A web service ω(ρ, ε, φ) is an atomic web service, where ρ is the precondition, ε is the effect and φ is its QoS.

- A task τ(ρ, ε, ϕ, η, WSτ,$\varsigma\omega$) is an abstract representation of all web services with similar functionality. That is the set of the web services which have the same precondition ρ and the same effect ε. The QoS of τ, $\phi = \min_{w\in\tau}(\varphi_w)$, η is the number of web service in τ, WSτ is the set of equivalent web services, and $\varsigma\omega$is the candidate web service to be selected with quality ϕ. T is the set of all tasks.

- A job J(ρ, ε, q, μ,Tj, $\varsigma\tau$) is the cluster of the tasks that have the same precondition ρ, the same effect ε. The QoS of J, $q = \min_{\tau\in J}(\phi_\tau)$, and finally, μ is the number of tasks in J, Tj is the set of tasks, and $\varsigma\tau$is the candidate task to be selected with quality q.

-A QoS Constraint Plan is a directed acyclic graph denoted as QCP [J(ρ, ε), E, Q] where: J(ρ, ε)={j_1 … j_n/j_i is a job i = 1, …, n} is the set of vertices representing the jobs in the graph where the precondition of j_1= ρ and the effect of j_n =ε. E ⊆ J X J is the set of edges. Q is the global QoS for the plan Q=$\sum_{j\epsilon QCP} q_j$. We assume that the jobs flow sequentially within QCP starting from j_1 until j_n.

-We define the Web Service Selection Problem (WSSP) as follow: given a QoS Constraint Plan QCP, the WSSP is to find the optimal web service ω for each j ϵ J that has the best quality by satisfying the following conditions:

$$C(\omega) = min\{ C(\omega i) , \forall \omega i \epsilon j \} \quad \text{-1-}$$

$$ET(\omega) = min\{ ET(\omega i) , \forall \omega i \epsilon j \} \quad \text{-2-}$$

-The Execution plan (EP) is defined as the solution of WSSP. It is denoted as EP={ (j1, ω1), (j2, ω2), ……, (jn, ωn)} where:

a. {j1, j2, ….. , jn } ϵQCP.
b. $\forall (j_i, \omega_i) \epsilon EP \Rightarrow (\exists \tau_i \epsilon j_i)$and $(\omega_i \epsilon \tau_i)$.

2.2. TASK AND JOB CLUSTERING:

As it was stated, the repository is made up of two-level of abstractions. The jobs: where each jobincludes a set of tasks having the same preconditions and effects. The tasks: where each task is a generic abstraction representing a set of serviceswith similarpreconditions and effects. The clustering is a continuous process thatupdates the tasks and jobs repositories. The clustering is built from the ground-up.

Each newly discovered web service (ω) isinsertedinto the Web Service repository WS and the update is propagated accordingly to the Task repository and the Job repository. The update propagation takes into account the definition and theconstraints (precondition and effect) of the discovered web service.

The first step in the clustering process is to update the task repository. For each web service ω(ρ, ε, φ), the precondition (ρ) and effect (ε) constraints are extracted from the service description profile. Then, match the constraints with the available tasks in task repository and insert ω into its belonging task. If no task matches the service constraints a new task τ is added to the task repository and assigned precondition τ.ρ, quality τ.ϕand effect τ.εfromω profile. For each inserting ω, the QoS ϕ is revised as to keep τ.ϕwith the minimum of all web services within the task cluster.

Several QoS attributes are considered for a single web service. In such problems a utility function is used to evaluate the multi-attributes quality of a set of alternative web services. For this purpose, we use the multiple attribute decision making, in particular the Simple Additive weighting (SAW) method (Hwang & Yoon 1981) is used. The evaluation score is calculated based on multiplying the scaled value for the alternative web services with the importance weight of the quality attribute, then adds the product for all attributes. The SAW uses the following formula to evaluate the alternatives.

$$Q_{\omega} = \sum_{a=1}^{n} w_a \cdot x_{\omega,a} \qquad \text{-3-}$$

Where Q_{ω} is the evaluation score of the web service ω, w_a is the weight of the attribute a, and $x_{\omega,a}$ is the score of ω with respect to attribute a. The maximum the value of Q_{ω}means the minimum the quality attribute values, thus the better quality. The value of $x_{\omega,a}$ is calculated for each web service within the same task cluster τ for each quality attribute based on the following formula:

$$x_{\omega,a} = \frac{Qmax_{j,a} - q_{\omega}}{Qmax_{j,a} - Qmin_{j,a}} \qquad \text{-4-}$$

For task clustering process, we evaluate and rank the quality of each web service within the same task cluster based on the values of execution time ET(ω) and cost C(ω). These quality values are extracted from the OWL-S description file for each web service ω. Knowing that, OWL-S is Web Ontology Language for describing Semantic Web Services.

On the other hand, creating the Job repository is made up in two times. First, for each task τ, a job J is created. The job J takes the precondition τ.ρ, the quality τ.ϕ, the effect τ.ε, and the singleton {τ} in its task set. As two simple tasks cannot have the same precondition and effect, a job J may

have at most one simple task. Composite tasks may be added later based on the circumstances. Each job will be updated by generating composite tasks with the same constraints. That is, if a task τ fails to achieve the request, a new composite task is prepared to substitute the failed task. The values of the job quality q, the job length μ and the candidate task $\varsigma\tau$ are updated accordingly.

The quality of the job j.q is calculated using SAW method. Based on the quality τ.ϕandlength τ.ηof the task τ, we compute the SAWτ of each task. Then we compute the SAW of the job. The job quality j.q will equal the Maximum of all SAWτ values that reflect the minimum of the execution time and the cost resulting with the best available quality.

The quality τ.ϕand the length τ.ηfor a composite tasks in each job cluster are the aggregation of these values in each task. The aggregation values for the Cost (C) and Execution Time (ET) for a composite task are the summation of their respective values in each task involved as shown in equation (5, 6).

$$C(\tau) = \sum_{\omega \in \tau} C(\omega_i) \qquad -5-$$

$$ET(\tau) = \sum_{\omega \in \tau} ET(\omega_i) \qquad -6-$$

2.3. CLUSTERING-BASED WEB SERVICE SELECTION:

As defined in section 2.1, the WSSP aims at finding the optimal web service to compensate each job in QCP plan. This optimality is achieved by finding the web services with the best Quality values. In our model, we utilize the two QoS attributes (Execution time, Cost) to direct the selection process. In the clustering process, each service ω is inserted in a task cluster τ. Furthermore, the web service with minimum quality is identified in the cluster as the candidate web service to be chosen (τ. $\varsigma\omega$). This process is also applied to the job repository for each new task inserted to any job cluster by identifying (J.ςτ).

By preparing the candidate web service (τ. $\varsigma\omega$) for each task, the Clustering-based Selection of web service as described in algorithm 1 is based on substituting each job in the QCP with its candidate task ςτ; then, substitute each task with its candidate web service $\varsigma\omega$. The output of the selection process is the execution plan (EP) as defined in section 2.1.

Algorithm 1: Clustering-based Selection

1: Procedure Clustering-based Selection **(**CP)

2:**Foreach**J ϵ CP**Do**
3: Selectτ ∈ J**Where** τ.ϕ=$\underset{\tau \in J}{MAX}$ (SAW)
4: **If**τ.getLength() =1**Then**
5: Selectω ϵ τ**Where** ω.φ= $\underset{\omega \epsilon \tau}{MAX}$ (SAW)
6: EP.Add (ω, J)
7: **EndIf**
8: **If**τ.getLength() > 1 **Then**

9: suppose ωc :composite empty task
10: **Foreach(**τk ∈ τ) **Do**
11: Selectω ϵ τk**Where** ω.φ=$\underset{\omega \epsilon \tau}{MAX}$ (SAW)
12: ωc.Append(ω')
13:**EndForeach**
14: EP**.**Add (ωc, J)
15: **EndIf**
16: **EndForeach**
17: **Return** EP

2.4 SELF-HEALING SELECTION ALGORITHM

The self-healing selection algorithm has the ability to recover any changes or faults that occur during selection phase of service composition process. It aims at substituting the faulty web service with similar web service from the same job/task cluster. Algorithm 2 describes the self-healing process; In case of web service failure, the algorithm test if the failed service belongs to an atomic task, if so, the failed service is replaced with an alternative one from that task. In the other hand, if the failed service is within a composite task, then replace it with the next best alternative web services based on its quality, and then recalculate the composite task quality. The next step is to determine if the composite task still has the minimum quality comparing with the tasks in its job cluster, then it will be substituted by job in EP. Else select an alternative task from the same job with the minimum task quality.

Algorithm 2: Self-healing Selection

1: Procedure Self-healing Selection(EP, ω)

2: τ<- ω.getTaskOwner()
3: foundAlternative<- False .
4: **If**τ.getLength() > 1 **Then**
5: Re-Calculate τ.ϕ
6: Select ω' ϵ τ . WSτwith ω'.ϕ= $\underset{\omega \ \in \tau.WS_\tau}{MAX}$ (SAW)
7: EP.Substitute(ω;ω')
8: foundAlternative<- True
9: **EndIf**
10: **If**τ.getLength() =1 **Then**
11: j <- τ.getJobOwner()
12: Re-Calculate j.q
13: Select τ' ϵ j . $Tj$with τ'.ϕ= $\underset{\tau \ \in j.Tj}{MAX}$ (SAW)
14: suppose ωc :composite empty task
15: **Foreach**(τk ϵ τ') **Do**
16: Select ω' ϵ τk . WSτwith ω'.ϕ= $\underset{\omega \ \in \tau k.WS_\tau}{MAX}$ (SAW)
17: ωc.Append(ω')
18: **EndForeach**
19: EP.Substitute(ω;ωc)
20: foundAlternative<- True
21: **EndIf**
22: **Return**foundAlternative

Example 1:consider the following clusters available in the task repository(table 1) with their QoS values. Each web service has two quality values, the first is the execution time ET measured in *ms*and the second is the cost C measured in $:

Table 1: Web Service Repository Example

WS							
ω1(a,b) (45)(27.2)	ω2(a,b) (71.75)(14.6)	ω3(a,b) (117)(23.4)	ω4(a,b) (105.2)(18)	ω5(a,b) (99.2)(13)	ω6(a,b) (108.2)(16.8)		
ω1(d,e) (105.4)(16)	ω2(d,e) (122)(15.4)	ω3(d,e) (96)(22)	ω4(d,e) (77.86)(7.3)				
ω1(c,d) (122)(15.4)	ω2(c,d) (114)(27)	ω3(c,d) (50)(2.8)	ω4(c,d) (126.2)(12)	ω5(c,d) (111)(26.9)	ω6(c,d) (75)(15)	ω7(c,d) (109.09)(8)	ω8(c,d) (125.75)(12.4)
ω1(a,c) (166.11)(10.3)	ω2(a,c) (169.33)(10.7)	ω3(a,c) (134)(11)					
ω1(d,a) (77.5)(17.5)	ω2(d,a) (119)(15.3)	ω3(d,a) (122.12)(13.5)	ω4(d,a) (113.5)(3)	ω5(d,a) (134.34)(2.1)			
ω1(b,d) (197.8)(23)	ω2(b,d) (158.17)(10.9						
ω1(d,e) (110)(15.4)	ω2(d,e) (133.4)(9.9)	ω3(d,e) (80.6)(11)	ω4(d,e) (150)(13)	ω5(d,e) (203.3)(12.4)			
ω1(d,f) (230.9)(22)	ω2(d,f) (140)(25.4)						
ω1(e,c) (80.3)(19)	ω1(e,c) (119.5)(15.6)	ω3(e,c) (148)(12.3)	ω4(e,c) (92.8)(14.2)				
ω1(f,e) (204.1)(18)	ω2(f,e) (187.6)(32.6)	ω3(f,e) (164)(23.7)	ω4(f,e) (90)(10)	ω5(f,e) (193.3)(15)	ω6(f,e) (136.8)(14.3)		

For each web service, we apply the SAW formula to calculate the normalized quality values. By assigning, for example, the weights of the criteria as W (ET)= 0.5 and W (C)= 0.3 meaning that the ET is more important than C. However the weight of a quality criterion could be adjusted according to the user preference. Based on this assumption, we can calculate the quality value for ω3 (a,b) as follow:

$$Q_{\omega 3} = \sum_{a=1}^{2} w_a \cdot \frac{Qmax_{j,a} - q_{\omega 3}}{Qmax_{j,a} - Qmin_{j,a}}$$

$$Q_{\omega 3} = w_{ET} \cdot \frac{Qmax_{j,ET} - q_{\omega 3}}{Qmax_{j,ET} - Qmin_{j,ET}} + w_C \cdot \frac{Qmax_{j,C} - q_{\omega 3}}{Qmax_{j,C} - Qmin_{j,C}}$$

$$Q_{\omega 3} = 0.5 \cdot \frac{117 - 117}{117 - 45} + 0.3 \cdot \frac{27.2 - 23.4}{27.2 - 14.6}$$

$$Q_{\omega 3} = 0 + 0.09 = 0.09$$

And accordingly, the quality values for the web services in each task cluster in the task repository will be as shown in Table 2

Table 2: Task Repository Example Including QoS

Task	τ.WS								τ.ϕ	τ.ςω	τ.η
τ1(a,b)	ω1(a,b) (0.5)	ω2(a,b) (0.61)	ω3(a,b) (0.09)	ω4(a,b) (0.301)	ω5(a,b) (0.46)	ω6(a,b) (0.309)			0.61	ω2	6
τ2(d,e)	ω1(d,e) (0.31)	ω2(d,e) (0.134)	ω3(d,e) (0.29)	ω4(d,e) (0.8)					0.8	ω4	4
τ3(c,d)	ω1(c,d) (0.17)	ω2(c,d) (0.08)	ω3(c,d) (0.8)	ω4(c,d) (0.18)	ω5(c,d) (0.10)	ω6(c,d) (0.48)	ω7(c,d) (0.34)	ω8(c,d) (0.184)	0.8	ω3	8
τ4(a,c)	ω1(a,c) (0.34)	ω2(a,c) (0.13)	ω3(a,c) (0.5)						0.5	ω3	3
τ5(d,a)	ω1(d,a) (0.5)	ω2(d,a) (0.18)	ω3(d,a) (0.19)	ω4(d,a) (0.47)	ω5(d,a) (0.3)				0.5	ω1	5
τ6(b,d)	ω1(b,d) (0)	ω2(b,d) (0.8)							0.8	ω2	2
τ7(d,e)	ω1(d,e) (0.38)	ω2(d,e) (0.58)	ω3(d,e) (0.74)	ω4(d,e) (0.34)	ω5(d,e) (0.16)				0.58	ω2	5
τ8(d,f)	ω1(d,f) (0)	ω2(d,f) (0.8)							0.8	ω2	2
τ9(e,c)	ω1(e,c) (0.5)	ω1(e,c) (0.36)	ω3(e,c) (0.3)	ω4(e,c) (0.62)					0.62	ω4	4
τ10(f,e)	ω1(f,e) (0.19)	ω2(f,e) (0.07)	ω3(f,e) (0.29)	ω4(f,e) (0.8)	ω5(f,e) (0.28)	ω6(f,e) (0.53)			0.8	ω4	6

Accordingly, The quality values for the tasks in the job repository is computed using eq. (5, 6), knowing that a task τ has two attributes as ranking criteria τ.ϕ, and τ.η. For example to calculate the quality of (τ1 τ5) in J2 (Table 3), we will apply eq. (5, 6), taking W(τ.ϕ)= 0.5 and W(τ.η) = 0.3:

$$Q_{\tau 1\tau 5} = \sum_{a=1}^{2} w_a \cdot \frac{Qmax_{\,j,a} - q_{\tau 1\tau 5}}{Qmax_{\,j,a} - Qmin_{\,j,a}}$$

$$Q_{\tau 1\tau 5} = w_{\tau.\phi} \cdot \frac{Qmax_{\,j,\tau.\phi} - q_{T1T5}}{Qmax_{\,j,\tau.\phi} - Qmin_{\,j,\tau.\phi}} + w_{\tau.\eta} \cdot \frac{Qmax_{\,j,\tau.\eta} - L_{\tau 1\tau 5}}{Qmax_{\,j,\tau.\eta} - Qmin_{\,j,\tau.\eta}}$$

Since ➔ $\tau.\phi_{\tau 1\tau 5} = \tau.\phi_{\tau 1} + \tau.\phi_{\tau 5} = 0.61 + 0.5 = 1.11$

And ➔ $\tau.\eta_{\tau 1\tau 5} = \tau.\eta_{\tau 1} + \tau.\eta_{\tau 5} = 6 + 5 = 11$

Then ➔ $Q_{\tau 1\tau 5} = 0.5 \cdot \frac{2.1-1.11}{2.1-0.8} + 0.3 \cdot \frac{13-11}{13-4}$

$$Q_{\tau 1\tau 5} = 0.38 + 0.06 = 0.44$$

Table 3 shows the job repository results after applying SAW formula.

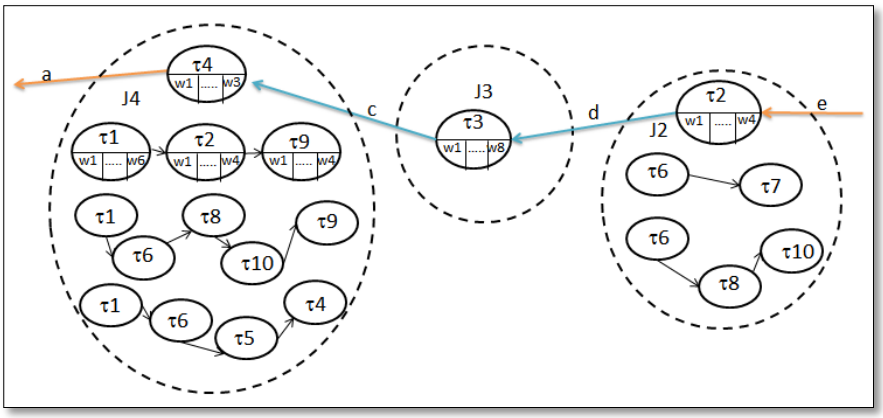
Table 3: Job Repository Example

Job	J.TJ			J.q	ςτ
J1(a,b)	τ1 (0)			0	τ1
J2(d,e)	τ2 (0.8)	τ6τ7 (0.44)	τ6 τ8τ10 (0)	0.8	τ2
J3(c,d)	τ3 (0)			0	τ3
J4(a,c)	τ4 (0.8)	τ1 τ2 τ9 (0.40)	τ1 τ6τ8 τ10τ9 (0)	τ1τ6 τ5τ4 (0.44)	0.8	τ4
J5 (d,a)	τ5 (0)			0	τ5
J6(b,d)	τ6 (0.8)	τ2 τ9 τ3 (0)		0.8	τ6
J7(d,e)	τ7 (0.8)	τ8 τ10 (0.4)	τ5 τ1 τ2 (0)	0.8	τ7
J8(d,f)	τ8 (0)			0	τ8
J9(e,c)	τ9 (0)			0	τ9
J10(f,e)	τ10 (0)			0	τ10

Example 2:

Consider the followingCPthat represents a plan with a precondition (a) and an effect (e) generated according to the jobs in previous table:

Figure 1: Quality Constraint Plan Example

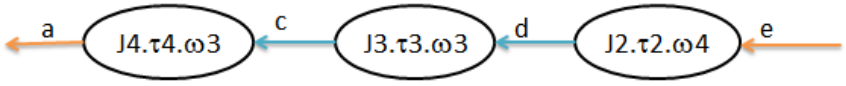


From the job and task repositories we can retrieve the following information:

Table 4: Service Selection Example

Job	ςτ	ςω
J4	τ4	ω3
J3	τ3	ω3
J2	τ2	ω4

According to table 4, the Execution Plan (EP) will be:



To illustrate the selection algorithm in case of service failure, we consider several situations; each example is independent of the other:

Situation 1:Suppose we have an event occurring in service repository that deletes J4.τ2.ω4. In this case, the deleted web service does not affect the selection process and nothing changed since ω4 is not the candidate service in task cluster τ2.

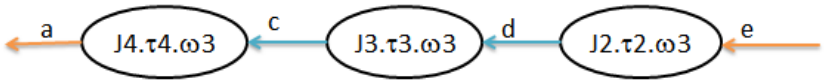
Situation 2:Suppose we have an event occurring in service repository that deletes the web services J2.τ2.ω1 and J2.τ2.ω4. In this case, the selection is affected and the algorithm will first recalculate the SAW values for the remaining web services in J2.τ2.

τ2 (b, e) {ω2 (b, e).ϕ=0.3 ω3 (b, e).ϕ = 0.5} τ2.η = 2

Then, recalculate the SAW values for the tasks in J2 and select the web service with maximum quality value.

J2 {τ2.ϕ=0.8 (τ1 τ5).ϕ= 0.3 (τ6 τ4 τ3).ϕ=0}

The web service with maximum quality value= J2.τ2.ω3. In this situation the Execution plan (EP) will result with:



2.5. TIME AND SPACE COMPLEXITY:

Lemma: For a web service failure within a job J in a given CP, the time complexity of the *Self-healing* Selection algorithm depends on the number of web services in the job J.

The recovery process aims at substituting the failed web service with a valid one from the same job. Therefore, the time needed to recovery is the time to traverse all the web services in the job to recalculate the quality, thus selecting the optimal service.

The number of services in a job is equal to all the services within the atomic and composite tasks. Accordingly, the number of services in a single composite task τ_h(that composed of *nh* simple tasks) and each simple task (τh_i) has nh_i web services= $\sum_{i=1}^{n_h} nh_i$

Thus, total number of web services in the job (where m is the number of tasks in j) is

$$nbServices = \sum_{h=1}^{m} \sum_{i=1}^{n_h} nh_i$$

3. PERFORMANCE EVALUATION:

3.1. DATASET AND EXPERIMENT SETUP:

In our experiment, we utilized several web service description files datasets to test our model, The OWLS-TC version 4.0 dataset, OPOSSum database for service descriptions [8], and SWS-TC-1.1 dataset provided by [9]. The OWLS-TC dataset provides 1083 semantic Web services written in OWL-S 1.1 from nine different domains (education, medical care, food, travel, communication, economy, weapons, geography and simulation)[10]. The OPOSSum is an open source project to collect semantic web services descriptions launched by Friedrich Schiller University in Germany, the data collection include 1129 OWL-S file in different domains. The SWS-TC-1.1 is a public dataset that contains 241 semantic web services descriptions in OWL-S also. Moreover, we collect extra web service descriptions from the web in order to raise the number of services and we came up to 6785 OWL-S services description files. After performing the job cluster algorithm we came up with 739 job clusters in the job repository and the same number in task repository.

However, the OWLS-TC dataset does not contain any information of the Quality values of the web services. Therefore we generate random values of Execution Time and Execution Cost that are picked from the QWS DATASET 1.0[11]. In our model we used only the Execution time and Cost quality parameters. Though, the model can be easily generalized to include other QoS parameters. The parser in our model extracted the main services parameters needed to the composition process from OWLS-TC dataset such as service name, input, output, preconditions, effect, quality values, and service description.

In order to evaluate the performance of the clustering-based selection and the self-healing algorithm we implementedtwoother algorithms that react to changes occurred in dynamic repository during selection phase.The algorithm provided by Barakat in [5] that stores the optimal path from the initial to the goal in each node in the plan.The traditional method of re-composition in case of faults occurs.

3.2. EXPERIMENTS RESULTS:

The experiment involves 6785 OWL-S files in the service repository, and after performing the job cluster algorithm we came up with 739 job clustersin the job repository. We performed two experiments in order to measure the performance of the selection and of the recovery in case of service fail.

3.2.1. Clustering based selection time:

In this experiment, we apply 100 test cases to measure the selection time of our method. For each test case the input is a composition plan (CP) that consists of jobs and the output is an execution plan (EP) that involves of the candidate web services. The plans vary in length; the range in our experiment start with plans consisting of 50jobs to plans containing500 jobs. Figure 2 shows the selection time resulted from the experiment of the clustering-based method. The time needed to select web services for each job within a plan is between 0.9 and 2.6 seconds.

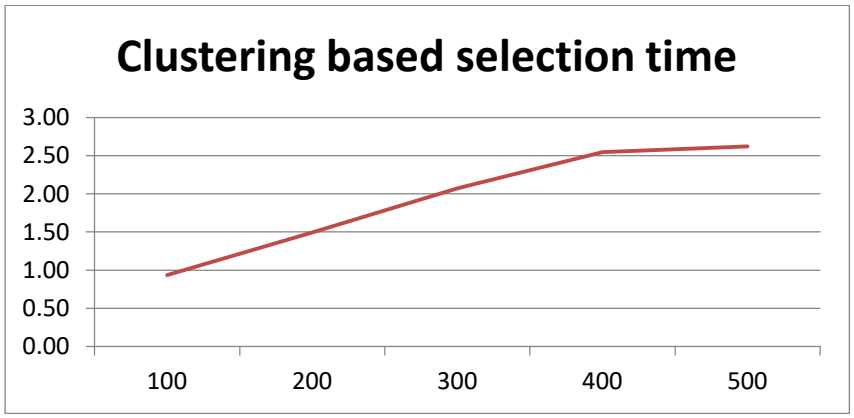


Figure 2: Clustering-based Selection Time

To highlightthe performance of the developed selection methodwith respect to the existing methods, we compared the performance of the clustering-based and barakat methods. Figure 3 shows the major enhancement in the selection time by the clustering-based selection. As one can notice, the execution time in barakat algorithm is much sloweras it requires storing all the paths from an exact node in the plan until the goal node. However, in our method, preparing in advance, the candidate services in each task cluster decreases considerably the time complexity of the selection process as shown in section 2.5.

Figure 3: Selection Time Comparison

3.2.2 Self-healing Recovery Time:

For this experiment, we picked a random set of plans and executed 100 test cases over the three methods: self-healing, barakat and reselection algorithms. For each test case, we performed the following:

- A random plan is entered to the selection algorithm.
- Two types of web service faults are simulated:
  - The first is the case when the faulty web services are recovered directly from either task or job repository. In this type, we performed three failed services; one that occurred in the

beginning of the plan, another, when a fault occurred in the middle of the plan and a third, when the fault occurred in the end of the plan.

By applying the input and simulation of several web service faults described previously, we ended up with 600 test cases to measure the time needed from the self-healing algorithm to recover a web service fault during selection phase.For each type of fault simulation, we calculated the average time needed to recover based on the number of services within the plan. The evaluation procedure is based on the number of the simulated faults. For each composite service length (50 -500), we applied first 10 faults during selection, then in the next step we increase the number by 10 until we simulated 60 faults. We will show the performance for the algorithms for each number of simulated faults (10 to 60).

The result of our experiment is shown in figure 5.It compares the three methods together.Figure 5 shows that the self-healing algorithm outperforms the other methods in the recovery time in selecting a substitute service for a faulty one. By preparing the tasks clusters and jobs clusters prior the selection time, the recovery time will cost the time needed to retrieve the job cluster where the fault occur and pick the service within that cluster that has the minimum quality values.

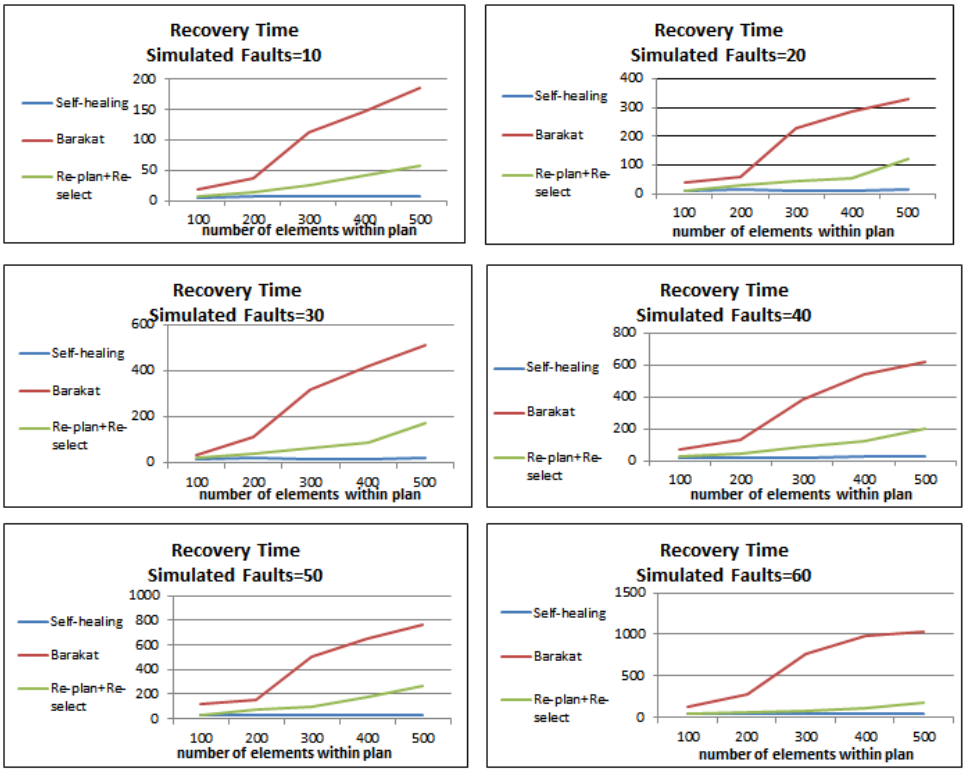


Figure 4:Recovery Time Comparison

4. CONCLUSION:

In this paper, we proposed a web service selection algorithm and a self-healing algorithm for service composition that react to changes in dynamic repository. The algorithm consists of three phases: first, continuous clustering of the similar web services to tasks and similar tasks to jobs. Second, a selection phase that chooses the most candidate web service from the task cluster to replace it in the plan. Third, a self-healing selection algorithm that reacts to changes in the web service repository by substituting the faulty service with similar one within the same cluster. An experiment is performed to test the selecting and recovering time for the proposed algorithm.

Moreover, a comparison with similar studies is done that shows an improvement in recovery time resulted from the proposed clustering algorithm.

REFERENCES:

[1] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," in Proceedings of the First international conference on Semantic Web Services and Web Process Composition, 2004, vol. 3387, pp. 43–54.

[2] M. Moghaddam and J. G. Davis, "Service Selection in Web Service Composition: A Comparative Review of Existing Approaches," in Web Services Foundations, First., A. Bouguettaya, Q. Z. Sheng, and F. Daniel, Eds. New York: Springer, 2014, pp. 321–346.

[3] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, "Exception Handling for Repair in Service-Based Processes," in IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2010, vol. 36, no. 2, pp. 198–216.

[4] K. Wiesner, R. Vacul, M. Kollingbaum, and K. Sycara, "Recovery Mechanisms for Semantic Web Services," in International Conference on Distributed applications and interoperable systems (DAIS), 2009, pp. 100–105.

[5] L. Barakat, S. Miles, and M. Luck, "Reactive Service Selection in Dynamic Service Environments," in First European Conference in Service-Oriented and Cloud Computing, 2012, pp. 17–31.

[6] M. Rajeswari, G. Sambasivam, N. Balaji, M. S. Saleem Basha, T. Vengattaraman, and P. Dhavachelvan, "Appraisal and analysis on various web service composition approaches based on QoS factors," J. King Saud Univ. - Comput. Inf. Sci., vol. 26, no. 1, pp. 143–152, Jan. 2014.

[7] C.-L. Hwang and K. Yoon, Multiple Attribute Decision Making: Methods and Applications A State-of-the- Art and Survey, 1st ed. Munich: Springer-Verlag Berlin Heidelberg, 1981.

[8] Opossum, "OPOSSum, an Online Portal for Semantic Services," FUSION group, Friedrich Schiller University Jena, Germany, 2010. [Online]. Available: http://fusion.cs.uni-jena.de/opossum/. [Accessed: 12-Feb-2017].

[9] Y. Ganjisaffar and H. Saboohi, "SWS-TC 1.1," SemWeb Central, 2006. [Online]. Available: http://projects.semwebcentral.org/projects/sws-tc/. [Accessed: 12-Feb-2017].

[10] M. Klusch and P. Kapahnke, "OWLS-TC," SemWebCentral, 2010. [Online]. Available: http://projects.semwebcentral.org. [Accessed: 01-Jan-2016].

[11] E. Al-Masri and Q. H. Mahmoud, "QoS-based Discovery and Ranking of Web Services," in IEEE 16th International Conference on Computer Communications and Networks (ICCCN), 2007, pp. 529–534.

AUTHORS

Prof. AmeurTouir, teaching in King Saud University in Riyadh KSU, he got his B.Sc. in computer science from Saint Etienne University in 1985. And his M.Sc. degreefrom Pierre & Marie Curie (Paris VI) University in Artificial Intelligence 1988.Ph.D. in Computer Science 1993 from Ecole Nationale Superieure des Telecommunications de Paris. Prof.Touirresearch interests is in Databases, spatial access methods, spatial query processing, spatio-temporal, object-oriented, design pattern.

Aram AlSedrani is a PHD student in King Saud University KSU in Riyadh, KSA. She got her B.Sc. degree from KSU in computer science in 1998, and got her M.Sc. degreefrom KSU in 2009. Aram research interest in software engineering, service oriented computing