Modern Applied Science; Vol. 12, No. 11; 2018
ISSN 1913-1844E-ISSN 1913-1852
Published by Canadian Center of Science and Education

Web Services Composition Using Dynamic Classification and Simulated Annealing

Issam AlHadid[1] & Evon Abu-Taieh[1]

[1] Faculty of Computer Information Systems, The University of Jordan, Aqaba, Jordan.

Correspondence: Issam AlHadid, Faculty of Computer Information Systems, The University of Jordan, Aqaba, Jordan. Tel: 962-79-9282636. E-mail: issamfe@hotmail.com

Received: April 15, 2018 Accepted: September 20, 2018 Online Published: October 29, 2018

doi:10.5539/mas.v12n11p395 URL: https://doi.org/10.5539/mas.v12n11p395

Abstract

Service Oriented Architecture (SOA) introduced the web services as distributed computing components that can be independently deployed and invoked by other services or software to provide simple or complex tasks. In this paper we propose a novel approach to solve the problem of the business processes execution engine web service selection and services composition in the Service Oriented Architecture (SOA) related to the Synchronous mode. The paper provides a mechanism to improve the web services selection and service composition, using dynamic web services and service composition classification and Simulated Annealing (SA) to satisfy services' requirements expressed as the Service Level Agreement (SLA). The results show that the proposed approach enhanced the services composition by increasing the availability and decreasing the response time to the service composite.

Keywords: Web Services, Service Composition, Service Level Agreement (SLA), Service Oriented Architecture (SOA), Classification, Simulated Annealing (SA).

1. Introduction

SOA introduced the web services as distributed computing components that collaborate with other services in a loosely coupled manner to perform simple tasks over the internet. Web Services are developed and deployed by different providers where the Web Service Definition Language (WSDL) is proposed as standard to avoid the interoperability and to expose the web service functions using eXtensible Markup Language (XML) and Simple Object Access Protocol (SOAP) to exchange data (Zhang et al., 2003; Juric et al., 2006; Zhang & Pan, 2008; Al Hadid, 2011).

Web services can be classified into two types; elementary web service and composite web service; while the elementary web service does not rely on other web service to accomplish its task. The composite web services are aggregating multiple web services which interact with each other according to a predefined business process model to perform an end-to-end complex functions using business process execution engine (Zeng et al., 2003; Gao et al., 2009; Sheng et al., 2014).

Business process execution engine supports asynchronous and synchronous processing modes. Synchronous mode is ideal when processes are executed in relatively short time where client is blocked until request processing is completed then it will return a response to the client immediately. On the other hand, asynchronous mode which is related to the long-running processes does not block the client for the duration of the operation (Juric et al., 2006; Li et al., 2010).

The Quality of Service (QoS) of service composition are influenced by the composed web services QoS which are the non-functional attributes related the Web service. non-functional attributes include service execution time, service availability, execution reliability, response time, service cost, where the value of the service composition QoS can be calculated using the QoS of each single web service component (Gao et al., 2009).

Web services operate and execute in a highly variable and dynamic environment (the web). As a result the Web services' QoS may evolve relatively either because of the internal evolution or because of external changes such as hosted environment changes or upgrade (Zeng et al., 2003). Also, web service provides some simple function that may not meet the requirements of complicated applications. There for, business process execution engines has been proposed to integrate web services into powerful composite value added service to provide effective and complex solution that meet the customers' needs (Gao et al., 2009). Consequently, statically approaches used to select web services and compose business processes in design time are inappropriate (Zeng et al., 2003); instead, dynamic composition approaches and techniques are required where the QoS changes of the composed web services during the runtime are taken into account.

In order to improve the web service selection and composition availability and reliability, some artificial intelligence (AI) algorithms have been adapted such Simulated Annealing (SA). SA is an AI widely used probabilistic search algorithm that can be used as a mechanism to simulates the behavior of metals under annealing when temperature is high, bad solutions have better chance to be accepted. As temperature goes down, SA becomes stricter and only good solution can be taken (Chau et al., 2008; Maqableh & Karajeh, 2014; Varty, 2017).

In this research we propose a new approach which is based on web services and composite services classification and SA algorithm. This approach solves the problems related to services composition and business process execution engines that support synchronous processing mode. Proposed approach improves the business process composition by classifying the web services and composite services using actions and weights. it also minimize the services composition time, support new composite services design and maintain existing composite services redesign. In addition, proposed approach improves web services reusability, utilization, providing highly dynamic composite services execution that meets the SLA. As a result, composite services will be updated to the WS QoS changes and to the different customers' requirements.

This paper is organized as follows: section 2 presents all the researches related to the web services selection and composition, in section 3, proposed approach is introduced, which includes the proposed execution engine architecture and the proposed execution algorithms. Section 4 discusses simulation and results, which includes the simulation configuration, experiments and the proposed dynamic composition approach results. finally, section 5 give the research conclusions.

2. Related Work

The significance of the Web Service selection and composition on-going research is an evident of the SOA and SLA aspects important to improve Web Services performance and reliability. Many Efforts have been addressed to identify the SLA attributes to improve the quality of service using heuristic and non-heuristic approaches (Mirzayi & Rafe, 2015).

Zhang et al. (2003) proposed a service selection mechanism used to configure and compose business processes using service selection mechanism applied to narrow down the available service list, and then utilize the optimization capabilities provided by the Genetic Algorithm to construct the best business processes that satisfy the customers' requirements. Jung et al. (2009) proposed methodology of business process clustering based on structural similarity metrics to compare business process models and to discovering the similar processes using the Cosine similarity measure. The researchers use the similarity metrics to classify similar processes in the same cluster utilized to reengineer and to support new process design. Li et al. (2010) proposed a distributed agent-based orchestration engine where agents collaborate to execute a portion of the original service composition. The implementation of the proposed architecture decreases the process execution time and improves the engine throughput compared to a non-distributed approach. Zhang & Pan (2008) build a web service classification system that uses different algorithms and classifiers to extract data from the corresponding WSDL file and parses it into key words related to the services groups. Corella & Castells (2006) proposed a heuristic approach for the semi-automatic classification of Web services using three-level matching procedure between services and classification categories, proposed approach assumed a corpus of classified services is available previously. Cardellini et al. (2017) present MOSES which is a software platform supporting QoS-driven adaptation of service-oriented systems, researchers claimed that proposed platform achieves a greater flexibility in facing different operating environments, also, it can handle any the possible QoS requirements conflicting related to several concurrent users.

Also, many researchers suggested Artificial Intelligence (AI) algorithms to improve the efficiency and effect of web services selection. Gao et al. (2009) proposed an algorithm named quality of experience/ quality of service driven simulated annealing and Genetic Algorithm to optimize the ability in web services selection. Wang and Hou (2008) proposed a web service selection algorithm based on multi-objective genetic by analyzing the aspects of the web service selection process constraints. Lécué (2009) proposed a Web Service Composition extensible optimization model designed to balance semantic similarities and quality of service QoS using Genetic Algorithms. Lei et al. (2005) presented dynamic selection of composite web services using genetic algorithm optimized neural network algorithm to express the composed service instead of using the traditional neural networks.

Many researchers stated that web services are affected by the performance and functional problems which may not guaranty the required Quality of Service (QoS) and the expected behavioral properties and functionality, where the traditional QoS evaluation methods might exhaust or slow down the service which may affect the business process and prevent the Service Level Agreement (SLA) to be achieved. Additionally, researchers claimed that current web services based on business process execution languages do not adequately satisfy the new service composition that meets the business requirements where new and the existing composite services must be redeployed dynamically to satisfy the clients' requirements which expressed as SLA, in addition to the problem of updating existing composite services according to the related web services QoS changes (Juric et al., 2006; Muthusamy et al., 2009).

3. Proposed Approach

3.1 Architecture

The Proposed architecture improves web services composition via dynamic classification in the business process execution engine initialization and runtime using SA to support new and existing process design and reengineer and to satisfy the clients' requirements which expressed as SLA. Service composition QoS is a key factor to ensure clients' satisfaction that may have different requirements regarding to the QoS. For example client may require maximizing the availability and response time while another may give more importance to the process than the response time. Accordingly, we have grouped the clients into three classes regarding the constraints and preferences of the users, also similar web services are grouped into classes using same actions and the weights. The classified Web services will be used to compose the services composition which will also be classified using actions and weights. Figure (1) illustrates the web services and services composition pools architecture.

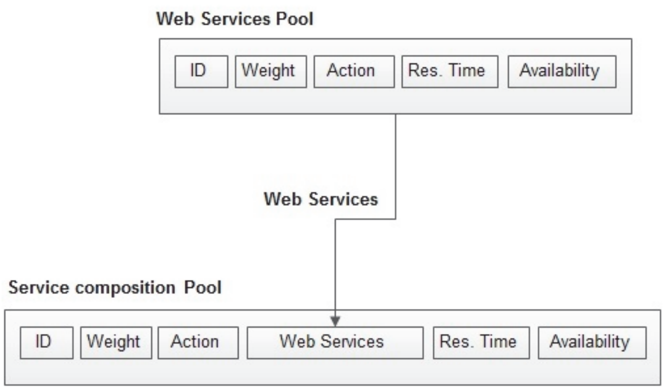

Figure 1. Web Services and Service Composition Pools Architecture

Web services are coupling components that collaborate with other services in a loosely coupled manner to perform complex tasks as new paradigm for distributed computing over the internet (Al Hadid, 2011; Afaneh & Al Hadid, 2013), Web Service Pool Architecture components include:

• Web service ID: represents a unique Web service identification number.

• Web Service Wight: demonstrates the QoS described by the availability and the response time of the web service, where web service's weight is classified into three categories; Excellent, Good and Poor service.

• Web Service Action: represents the affect of the web service process.

• Web service Response Time: is the time needed by a service to correctly respond to the request, it can be calculated using the following equation (Emeakaroha et al., 2010):

$$WSResTime = WSResTime(in) + WSResTime\ (out) \tag{1}$$

Where;

$$WSResTime(in) = \frac{packetsize}{availablebandwidthin-inbytes} \tag{2}$$

$$\mathrm{WSResTime(out)} = \frac{\mathrm{packetsize}}{\mathrm{availablebandwidthin-outbytes}} \tag{3}$$

• Web service Availability: is the probability that the web service is accessible; it can be calculated using the following equation (Emeakaroha et al., 2010):

$$WSava = \ 1 \ \frac{Down\ \text{Time}}{\text{Up Time}} \tag{4}$$

Where:

- Down time represents the time it takes to bring the web service back online after a failure, which is known as the mean time to repair (MTTR).
- Up time denotes the web service operational time between the last web service failure to the next failure, which is known as the mean time between failure (MTBF).

Service composition pool architecture components are:

• Service Composition ID: represents a unique Service Composition identification number.

• Web Services: Web services that composed the Service Composition which depends on action and weight of the related Web services.

• Service Composition Weight: demonstrates the QoS described by the availability and the response time of the Business Process, where business process weight is classified into three categories; Excellent, Good and Poor business process.

• Service Composition Action: Service Composition with same action has the same affect with different web services invocations. According to the SLA and QoS which can be recognized by the Weight (strength of classifier). Same business processes have the exact same action.

• Service Composition Response Time: is the total time of every web service execution time in the service composition. it can be calculated using the following equation (Gao et al., 2009):

$$BusProctime = \sum_{i=1}^{N} WStime(i) \tag{5}$$

Where,

- BusProctime is the sum of all the web services' response time.
- WStime(i) is the time needed by web service (i) to correctly response to the business process task request.
- N: is the number of web services in the service composition

Web services and service composition response time has been categorized as shown in table 1.

Table 1. Web Service and Service Composition Response Time (Ludwig et al., 2003; Gao et al., 2009)

Band	Value
Excellent response time	0.2-0.5 Sec.
Good response time	0.6-1.0 Sec.
Poor response time	1.1 - 3.0 Sec.

• service composition Availability can be obtained by calculating the multiplication of all web services availability composing the business process, as shown in the following equation (Gao et al., 2009):

$$BusProcava = \prod_{i=1}^{N} WSava(i) \tag{6}$$

Where,

- BusProcava is the multiplication of all the web services availability in the business process.
    - $WS_{ava}(i)$ is the probability that the web service (i) is accessible.
- N: is the number of web services in the service composition

Web services and service composition availability has been categorized as shown in table 2.

Table 2. Web Service and Service Composition Availability (Ludwig et al., 2003)

Band	Value
Excellent availability	99%-100%
Good availability	97% - 98%
Poor availability	94% - 96%

SLA also represented as a SLA pool which will improve the procedure of selecting the precise business process that achieves the client SLA requirements. SLA pool Architecture is shown in Figure (2).

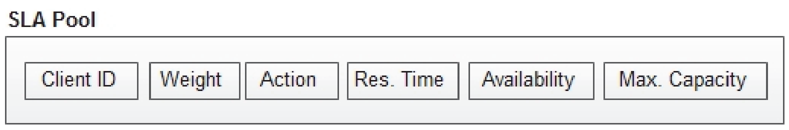

Figure 2. SLA Pool Architecture

SLA Pool Architecture components are:

• Client ID: represents a unique client identification number.

• Client SLA Weight: demonstrates the QoS described by the availability and the response time of the client requests, where the SLA weight is classified into three categories; Excellent, Good and Poor client's SLA.

• Client SLA Action: action represents the affect of the Business Process.

• Client SLA Response Time: the maximum response time of the composite service.

• Client SLA Availability: the minimum availability of the composite service.

• Client SLA Maximum Capacity: maximum number of client's requests a business process can accept per second.

The architecture of the proposed approach for the services composition using dynamic classification and SA
is shown in Figure 3.

3.2 Proposed Algorithm

Proposed approach simulation is modeled as a set of classified web services and composite services to provide the execution environment and the SLA aspects, each composite service is defined as a sequence of activities implemented by execution operations on the web services that have the same weight and related actions to fulfill the composite services requirements.

We will apply the proposed architecture to the following execution modes which will be compared with the results applied without the modifications of the suggested architecture. The execution modes are:

1) Normal Mode: Clients' requests within the SLA requests maximum capacity (Sheng et al., 2014).
2) Dropped Mode: Clients' requests exceed the SLA requests maximum capacity which considered as Denial of Service Attack (DoS) (Jung et al., 2002).
3) Priority Mode: Clients' requests within the SLA requests maximum. in this case, there will be more than one request will be received in the same time (Manikutty & Cao, 1998), and the proposed algorithm will arrange the invocations according to the clients' weight, and number of requests.

3.2.1 Initialization:

1) Load Clients SLA Data into SLA_Pool;
2) Load Web_Services_Pool data; web services will be classified into clusters using action and weight stored in the WSDL;
3) Load Business_Processes_Pool data by selecting the web services from the Web_Services_Pool that meet the service composition action and weight requirements randomly. Accordingly, selected web services' availability and response time data will be used to calculate the service composition availability and total response time of the associated web services.

3.2.2 Run Execution Engine using Normal Mode

In this mode, number of client's requests will not exceed the requests maximum capacity identified in the SLA

Pool for each client:

```
    Do until Run=No_Of_Runs
            Select client randomly from the SLA_Pool;
            Generate No_Of_Business_Process_requests the client will invoke, where
            No_Of_Service_Composition_requests <= Client_Max_Capacity;
            Do Until Client_Request = No_Of_ Service_Composition_requests
                    From the service composition pool that matches the clients SLA weight and action,
                    Select randomly an unengaged service composition and record its response time,
                    availability, weight;
            Loop (Next Client_Request)
            Calculate the Average of the client's requests availability and response time;
            Record Client, No_Of_ Service_Composition_requests, availability, response time in the
            results file;
            Generate Random_Number, where Random_Number between 0 and 100;
            If Random_Number > Last_Business_Process_Availability
                    Run SA GAP; reclassify web services into different pools using action and weight
                    stored in the WSDL and recreate the service composition according to the new web
                    services classification and update the web services and service composition pools;
            End if;
    Loop (Next Run)
```

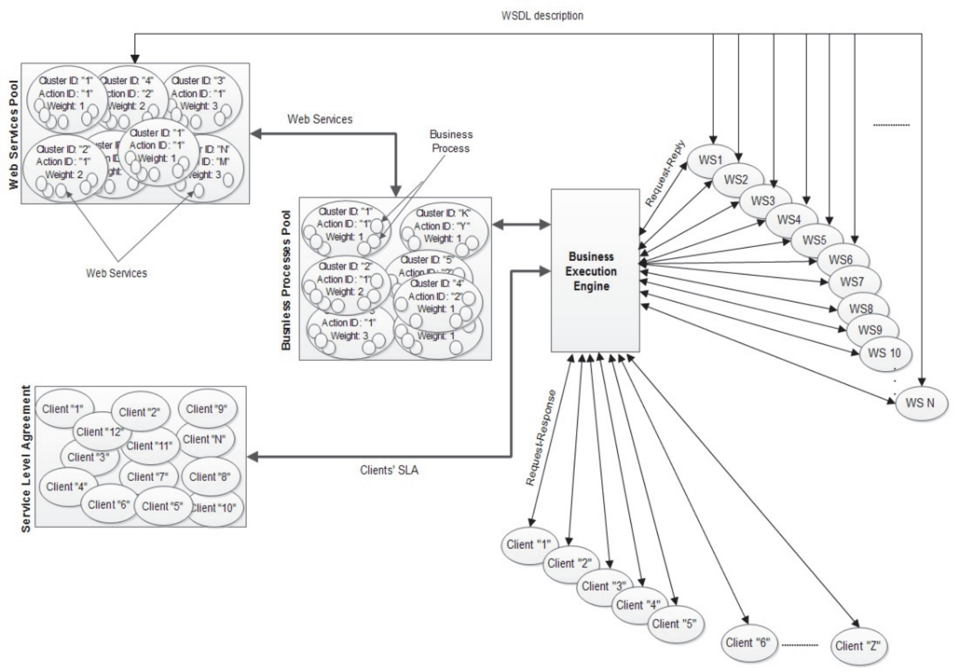

Figure 3. Architecture of the Proposed Approach for Services Composition Using Dynamic Classification and Simulated Annealing

3.2.3 Run Execution Engine using Dropped Mode

In this mode, the system will drop the number of client's requests that exceeds the requests maximum capacity identified in the SLA Pool for each client.

```
    Do until Run=No_Of_Runs
```

```
            Select client randomly from the SLA_Pool;
            Generate No_Of_Service_Composition_requests the client will invoke;
            If No_Of_Service_Composition_requests >Client_Max_Capacity then
                Dropped_requests = No_Of_Service_Composition_requests - Client_Max_Capacity
                No_Of_Service_Composition_requests = Client_Max_Capacity;
            End if;
            Do Until Client_Request = No_Of_Service_Composition_requests
                From the service composition pool that matches the clients SLA weight and action,
                Select randomly an unengaged service composition and record its response time,
                availability, weight;
            Loop (Next Client_Request)
            Calculate the Average of the client's requests availability and response time;
            Record Client, No_Of_Service_Composition_requests, availability, response time,
            Dropped_requests in the results file;
        Generate Random_Number, where Random_Number between 0 and 100;
            If Random_Number > Last_Service_Composition_Availability
                Run SA GAP; reclassify web services into different pools using Action and Weight
                stored in the WSDL and recreate the business process according to the new web
                services classification and update the web services and service composition pools;
            -End if;
        Loop (Next Run)
```

3.2.4 Run Execution Engine using Priority Mode

In this mode, the system will arrange the invocations according to the clients' weight, and number of requests:

```
        Do until Run=No_Of_Runs
         Do Until Priority_Client = No_of_Concurrent_Clients -1
                Select client randomly from the SLA_Pool;
                Generate No_Of_Service_Composition requests the client will invoke where
                No_Of_Service_Composition_requests <=Client_Max_Capacity;
                Load client_Number and No_Of_Service_Composition_requests to the Priority_Pool;
            Loop (Next_Priority_Clients)
        Arrange Priority_Pool data based on the Client_Weight and No_of_Service_Composition_requests;
        Do Until Priority_Client = No_of_Concurrent_Clients
        Do Until Client_Request = No_Of_Business_Process_requests
                From the business process pool that match the clients SLA weight and action,    Select
                randomly an unengaged service composition and record its response time, availability,
                weight;
                Loop (Next Client_Request)
                Calculate the Average of the client's requests availability and response time;
            Record Client, No_Of_Service_Composition_requests, availability, response time in the results
            file;
            Loop (Next_ Client_Request)
            Loop (Next_Priority_Clients)
            Generate Random Number (R), where (R) between 0 and 100;
            Generate Random Number (X), where (X) between 0 and No_of_Concurrent_Clients - 1, get
            the client's requests availability from the results file where the client number is located in the
            Priority_Pool in location (X);
            If (R) > client (X) availability
                  Run SA GAP; reclassify web services into different pools using Action and Weight
                  stored in the WSDL and recreate the service composition according to the new web
                  services classification and update the web services and service composition pools;
            End if;
      Loop (Next Run)
```

4. Simulation and Results

4.1 Simulation

In order to evaluate the proposed service selection and composition approach, we develop a special simulator for web services selection and composition using VB.NET. Proposed simulator supports the web services and service compositionclassification, SLA, web services and business process pools and SA, in addition to SLA Gap which simulates the case when the process availability or reliability do not meet the client SLA needs; SLA Gap is used to modify the Web services specifications by changing web services weight to upper or lower level and modify its availability and response time and update web services pool. Accordingly, composite services weight will be updated to the new level and adjust its availability and response time values according to the Table (1) and Table (2).

Table (3) shows the simulation parameters of the different modes and implementations. To compare and evaluate simulator results, some parameters of the simulation setup are common to all the modes such as number of web services, number of composite services and number of execution runs.

Simulator testing methods are grouped into two groups which applied to all business process execution modes; normal, dropped and priority modes. The first testing groups do not classify the web services or the service composition modes while the other testing group uses the proposed approach to dynamically classify the web services and composite services and SA.

Table 3. Simulator Set-Up Parameters

Parameter	Value
Number of web services	50
Number of business processes	15
Number of web services composed service composite	5
Number of Clients	10
Number of Runs (Normal and Dropped modes)	1000
Number of Runs (Priority mode)	250
Number of Concurrent Clients (Priority mode)	4
Service composite maximum capacity	1800
Excellent web service and service composite weight	1
Good web service and service composite weight	2
Poor web service and service composite weight	3

4.2 Results

This section discusses the results of the web services and composite services QoS obtained from the simulator experiments for the standard static approach and proposed dynamic service composition approach. We implemented the experiments 50 times for the standard and proposed improved approaches using the simulator with 1000 runs in each experiment for every normal, dropped and priority modes.

Table (4) shows the availability of the proposed and the standard service composition for all simulator experiments modes; normal, dropped and priority modes by calculating the average values of the composite services availability property during the simulator runs. In Table (4) we classified the availability results according to the SLA clients' groups; excellent, good and poor SLA.

Table 4. Composite services average availability for Normal, Dropped and Priority modes

Mode	Excellent Standard	Excellent Improved	Good Standard	Good Improved	Poor Standard	Poor Improved
Normal	99.31%	99.32%	97.710%	97.760%	94.798%	94.800%
Dropped	99.201%	99.761%	97.194%	97.920%	94.123%	94.819%
Priority	99.103%	99.420%	97.419%	97.830%	94.246%	94.750%

Figure (4) shows the average of composite services availability for Normal, Dropped and Priority modes, the Figure shows a comparison between the standard execution for the business process execution engine algorithm and the proposed improved algorithm. Figure (4) shows that the results of the proposed approach for all modes are better than the standard business process execution engines modes; where the composite services' availability is higher than the standard service composition approaches for all modes.

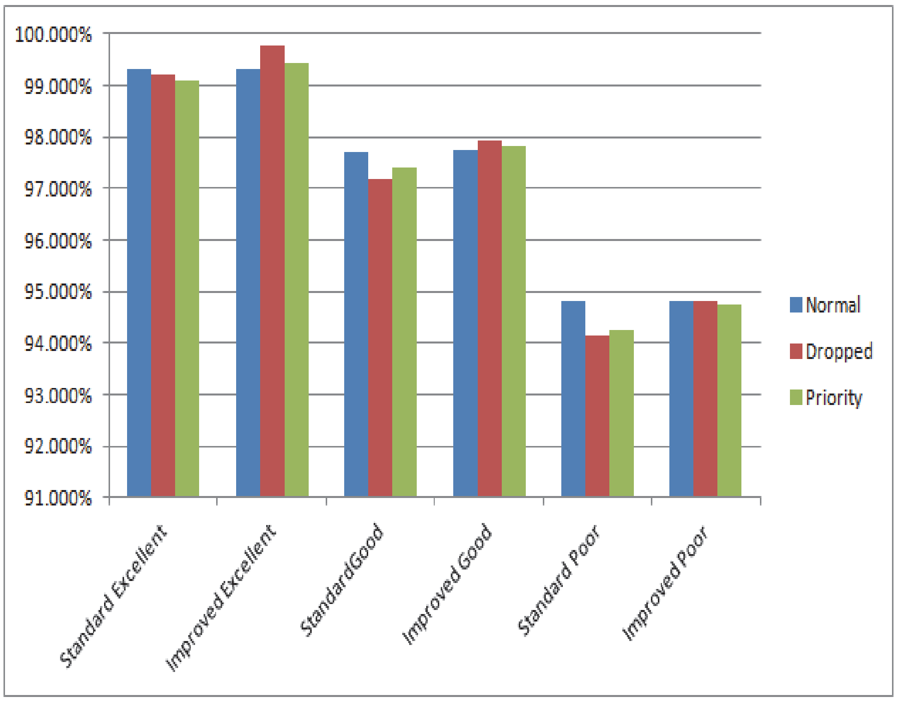

Figure 4. Average of business process availability for Normal, Dropped and Priority modes.

Table (5) shows the average values proposed and standard service composition response time related to all of the simulator experiments modes; normal, dropped and priority modes.

Table 5. Improved and Standard Service Composition Response Time for Normal, Dropped and Priority Modes

Mode	Excellent Standard	Excellent Improved	Good Standard	Good Improved	Poor Standard	Poor Improved
Normal	2.39 sec.	1.58 sec.	5.72 sec.	4.20 sec.	13.06 sec.	10.42 sec.
Dropped	2.406 sec.	1.570 sec.	5.695 sec.	4.199 sec.	13.050 sec.	10.419 sec.
Priority	2.300 sec.	1.619 sec.	5.208 sec.	4.099 sec.	12.195 sec.	10.219 sec.

Figure (5) shows that the average response time of the improved service composition using the proposed algorithm is better than the standard service composition approaches, where the response time of the proposed approach modes are less than the standard service composition modes.

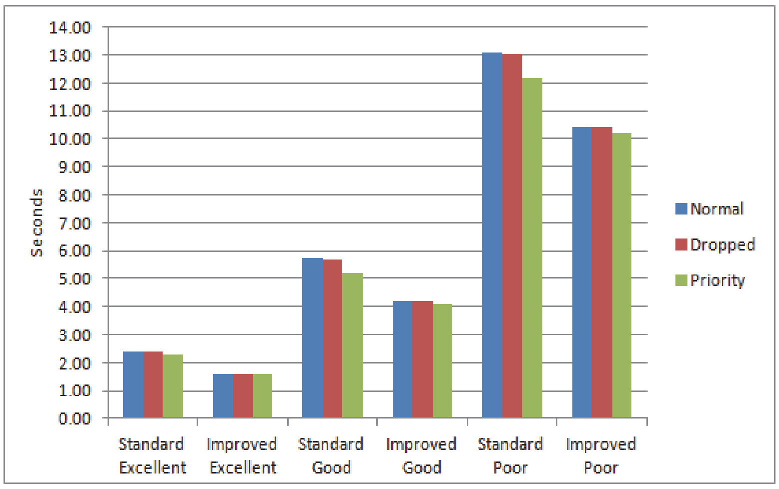

Figure 5. Average of composite services response time for Normal, Dropped and Priority modes

As shown in Figure (4) and Figure (5), the obtained QoS results show that proposed algorithm has enhanced service composition availability and response time for all modes and all the SLA clients' group: excellent, good and poor service groups, where the improved proposed approach has maximized the composite web services availability and minimized the composite web services response time based on the classification processes and SA Gap for all modes. Accordingly, proposed approach using classification and SA is the most appropriate choice to be used by the business process execution engines.

5. Conclusion

This paper presents a dynamic approach to classify, select and compose web services with QoS constraints. The proposed approach implements the classification pools and SA to dynamically classify the web services and composite services to the different pools based on the updated QoS in order to meet the clients' requirements and needs.We designed the proposed approach to update the web services QoS properties during the business execution engine run time which improves the engine performance by recomposing the services composition dynamically.

Our approach depends on SA algorithm to upgrade the composite services; it means that it will be updated randomly based on the service composition availability and the generated random number which give the engine the flexibility to update the web services and services composition pools. On the other hand, SA is activated and run when the generated random number is greater than the composite service availability. The service composition process might be trapped in local optimum problem, requiting a longer time to update the web services pool and the service composition pool because the value of the composite services availability is high even if the selected service composition related to a poor SLA class.

References

Afaneh, S., & Al Hadid, I. (2013). Airport enterprise service bus with three levels self-healing architecture (AESB-3LSH). *International Journal of Space Technology Management and Innovation (IJSTMI)*, *3*(2), 1-23.

Al Hadid, I. (2011). Airport enterprise service bus with self-healing architecture (aesb-sh). *International Journal of Aviation Technology, Engineering and Management (IJATEM)*, *1*(1), 1-13.

Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Presti, F. L., & Mirandola, R. (2017). MOSES: A platform for Eperimenting with QoS-driven self-adaptation policies for service oriented systems. In *Software Engineering for Self-Adaptive Systems III. Assurances* (pp. 409-433). Springer, Cham.

Chau, T., Muthusamy, V., Jacobsen, H. A., Litani, E., Chan, A., & Coulthard, P. (2008, October). Automating SLA modeling. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds* (p. 10). ACM.

Corella, M. Á., & Castells, P. (2006, September). Semi-automatic semantic-based web service classification. In *International Conference on Business Process Management* (pp. 459-470). Springer, Berlin, Heidelberg.

Emeakaroha, V. C., Brandic, I., Maurer, M., & Dustdar, S. (2010, June). Low level metrics to high level

SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on* (pp. 48-54). IEEE.

Gao, Z. P., Jian, C., Qiu, X. S., & Meng, L. M. (2009). QoE/QoS driven simulated annealing-based genetic algorithm for Web services selection. *The Journal of China Universities of Posts and Telecommunications*, *16*, 102-107.

Jung, J. Y., Bae, J., & Liu, L. (2009). Hierarchical clustering of business process models. *International Journal of Innovative Computing, Information and Control*, *5*(12), 1349-4198.

Jung, J., Krishnamurthy, B., & Rabinovich, M. (2002, May). Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the 11th International Conference on World Wide Web* (pp. 293-304). ACM.

Juric, M.B., Mathew, B., & Sarang, P.G. (2006). *Business process execution language for web services: An architect and developer's guide to orchestrating web services using BPEL4WS*. Packt Publishing Ltd.

Lécué, F. (2009, October). Optimizing qos-aware semantic web service composition. In *International Semantic Web Conference* (pp. 375-391). Springer, Berlin, Heidelberg.

Lei, Y. A. N.G., Yu, D. A. I., Zhang, B., & Yan, G. A. O. (2005, November). Dynamic selection of composite web services based on a genetic algorithm optimized new structured neural network. In *null* (pp. 515-522). IEEE.

Li, G., Muthusamy, V., & Jacobsen, H. A. (2010). A distributed service-oriented architecture for business process execution. *ACM Transactions on the Web (TWEB)*, *4*(1), 2.

Ludwig, H., Keller, A., Dan, A., King, R. P., & Franck, R. (2003). Web service level agreement (WSLA) language specification. *IBM Corporation*, 815-824.

Manikutty, J. A. M. D. A., & Cao, P. (1998, June). Providing differentiated levels of service in web content hosting. In *Proc. First Workshop Internet Server Performance*.

Maqableh, M., & Karajeh, H. (2014). Job scheduling for cloud computing using neural networks. *Communications and Network, 6*(3), 191-200.

Mirzayi, S., & Rafe, V. (2015). A hybrid heuristic workflow scheduling algorithm for cloud computing environments. *Journal of Experimental & Theoretical Artificial Intelligence*, *27*(6), 721-735.

Muthusamy, V., Jacobsen, H. A., Chau, T., Chan, A., & Coulthard, P. (2009, November). SLA-driven business process management in SOA. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research* (pp. 86-100). IBM Corp.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences*, *280*, 218-238.

Varty, Z. (2017). *Simulated Annealing Overview.* Retrieved from http://www.lancaster.ac.uk/~varty/RTOne.pdf

Wang, J., & Hou, Y. (2008, October). Optimal web service selection based on multi-objective genetic algorithm. *2008 International Symposium on Computational Intelligence and Design 2008 ISCID'08*, (Vol. 1, pp. 553-556), IEEE.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., & Sheng, Q. Z. (2003, May). Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web* (pp. 411-421). ACM.

Zhang, J., & Pan, D. (2008). Web service classification. *Dan Pan, Jing Zhang*.

Zhang, L. J., Li, B., Chao, T., & Chang, H. (2003, October). On demand web services-based business process composition. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on* (Vol. 4, pp. 4057-4064). IEEE.

Copyrights