Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2021.014892
Article

Tech Science Press

Optimizing Service Composition (SC) Using Smart Multistage Forward Search (SMFS)

Issam Alhadid[1], Hassan Tarawneh[2], Khalid Kaabneh[2], Ra'ed Masa'deh[3], Nawaf N. Hamadneh[4,*], Muhammad Tahir[5] and Sufian Khwaldeh[1]

[1]Faculty of Information Technology and Systems, University of Jordan, Aqaba, Jordan
[2]Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan
[3]School of Business, University of Jordan, Amman, Jordan
[4]Department of Basic Sciences, College of Science and Theoretical Studies, Saudi Electronic University, Riyadh 11673, Saudi Arabia
[5]Department of Computer Science, College of Computing and Informatics, Saudi Electronic University, Riyadh 11673, Saudi Arabia
*Corresponding Author: Nawaf N. Hamadneh. Email: nwwaf977@gmail.com
Received: 25 October 2020; Accepted: 15 February 2021

Abstract: Service Oriented Architecture (SOA) is a style of software design where Web Services (WS) provide services to the other components through a communication protocol over a network. WS components are managed, updated, and rearranged at runtime to provide the business processes as SCs, which consist of a set of WSs that can be invoked in a specific order to fulfill the clients' requests. According to the Service Level Agreement (SLA) requirements, WS selection and composition are significant perspectives of research to meet the clients' expectations. This paper presents an effective technique using SMFS that attempts to improve the WS selection as well as SC construction and ultimately optimize the WS resource utilization. The results show that the proposed SMFS technique enhances the WS resource utilization by 9.6% compared to the standard Multistage Forward Search (MFS) technique. Similarly, the number of constructed SCs using the proposed SMFS technique are increased by 36.97% compared to the number of constructed SCs with the standard MFS technique.

Keywords: Service oriented architecture; service composition; smart multistage forward search; machine learning; optimization algorithms

1 Introduction

SOA is a style of software design to connect different applications and technologies using the WS components. An automatic adaptation can be effectively implemented in the systems of SOA using machine learning algorithms [1]. WSs are independent, distributed, loosely coupled, and reusable software components that encapsulate a discrete functionality. WSs can be deployed and invoked by other WSs or software to perform simple or complex tasks using standard internet and eXtensible Mark-up Language (XML) based protocols [2–5].

WS technology has been adopted by different enterprises to implement and provide business process workflows, which is referred to as SC. Each SC consists of different WSs, which can be replaced, managed, and updated at runtime without interrupting the ongoing business processes [6–8]. Due to the highly dynamic nature of the Internet, many factors affect the WS Quality of Service (QoS). The internal evolution as well as external modifications related to the hosted environment and network have a direct impact on the WS QoS [8]. Some additional factors that might affect the QoS of WSs include the WS poor performance, scalability, handling a massive number of synchronous requests to guarantee SLA requirements, expected functionality, and behavior [3,9]. Dynamic as well as automatic WS selection and composition with optimal QoS attributes are still complex processes that are considered challenging to satisfy the customers' SLA functional and non-functional requirements [10–12].

Issues related to WS QoS modifications, SC reconstruction, and WS resource utilization are discussed in [3,13,14]. In this regard, Fan et al. [15] claimed that the composition approaches are not efficient enough when applied in real-time and large-scale environments. The authors stated that finding the minimum number of WSs with identical and similar functionality to create the SC on a large-scale is not an easy process. Researchers argue that minimizing the number of WSs not only improves the maintenance and management of SCs but also increases the success rate of responses, saves resources, and minimizes cost. According to Al-Hadid and Abu-Taieh [3], each WS has limited capacity such as the number of maximum requests that can be accepted each second. Many researchers argue that the acceptance of maximum requests per second by SCs depends on the integrated WSs and the maximum capacity of each WS. It should be noted that the maximum capacity of a WS may not be the same for all integrated WSs in a SC because the maximum capacity of a SC depends on the minimum value of a WS capacity in a composition [3]. This signifies that the WS with greater capacity value will acquire the available resources not used by the SC. Al-Hadid and Abu-Taieh [3] further claimed that the available resources of a WS are not utilized to their optimum capacity in the construction process of a SC. In this connection, they suggested to select WSs with almost the same maximum capacity for optimum utilization of WS resources. The literature review guided us towards sound analysis of four major research perspectives as listed below.

- Adopting a dynamic approach to improve the Orchestrator processes of selecting and creating SCs.
- Upgrading the composite services according to the QoS attributes.
- Enhancing the SC scalability.
- Optimizing the utilization of WS resources in a SC process.

Consequently, we propose an efficient technique based on SMFS for the solution of SC problem. Our key contributions include:

- Transforming the WS repository into a dependency service graph.
- Employing SMFS algorithm to find the path from source to destination, which represents the best SC selection and construction with minimum cost.
- Optimizing the utilization of WS resources by acquiring the unused and available WS resources to construct new SCs.

The experimental results of the proposed SMFS technique demonstrated superior performance compared to other techniques and mechanisms in terms of efficiency and quality in optimizing the utilization of WS resources. However, it needs additional time for calculation, reuse, and utilization of WS available resources while constructing new SCs after the first composition phase.

This research is organized as follows. Section 2 presents an overview of the related literature about WS selection and composition. Section 3, describes the proposed technique, which covers modification to the

execution engine architecture and SMFS technique. Section 4 discusses the dataset, results and analysis. Section 5 gives the research conclusions and the future directions.

2 Related Work

WS selection and composition has been addressed as one of the active research areas to improve its performance and reliability using different heuristic and non-heuristic approaches [7,16]. In order to achieve the SLA requirements, some researchers suggest to ignore the clients' requests once the SLA maximum capacity is exceeded in order to avoid any Denial of Service (DoS) and the Distributed Denial of Service (DDoS) attacks [17,18] whereas some others recommend to prioritize the clients' requests according to the clients' class [3]. In this connection, Al Hadid and Abu-Taieh [3] proposed a Simulated Annealing (SA) based dynamic mechanism to enhance the WS selection and SC process to achieve the SLA requirements as well as improve the SC availability and response time. Gao et al. [19] adopted a dynamic programming based approach in order to generate a weighted multistage graph where the longest path in the generated graph represents the solution to the WS selection and composition problem. In the context of SC problem, metaheuristic optimization algorithms have also been utilized [20]. In this regard, Shree et al. [21] proposed a method using integrated Ant Colony Optimization Algorithm coupled with Artificial Bee Colony Optimization Algorithm (IACO-ABCOA) to find the optimal WS configuration solution for solving stagnation and convergence problems. Moreover, researchers argue that IACO-ABCOA method can be used in the directed workflow model as a directed acyclic graph to determine the optimal feasible path that represents the best SC. Jung et al. [22] proposed cosine similarity based method for business process clustering by identifying similar processes to support new business process designs. Gao et al. [23] developed SA and Genetic Algorithm based technique to optimize the WS selection process. Elmaghraoui et al. [24] modeled semantic relationships of all WSs as a directed graph to find all shortest paths to optimize the computational efforts associated with WS composition. Fan et al. [15] proposed a new technique to calculate minimum composition that satisfies the clients' SLA requirements. Using the relevant WS, the proposed mechanism generates a service dependency graph and then transforms each search step into dynamic knapsack problem. These relevant WSs are then mapped to items with changeable volume and cost.

The cost structure of using WSs have been discussed by many researchers. In this connection, Lin et al. [25] stated that WSs cost is not universally structured however, there exist two pricing models for pricing the WSs namely profit maximization and welfare maximization. Profit maximization structure is adopted by WS providers to achieve maximum benefits without considering the volume of WSs used by clients whereas welfare maximization structure is used to set prices of WSs by managing the best trade-off between own profit and clients' utilization. Another pricing framework is developed by Mathew et al. [26] on the basis of different factors such as QoS, cost of service, and the volume of clients' transactions. The framework suggests three pricing models, which are:

1. Subscription-Based Pricing for Commoditized WSs where customers are charged for unlimited use over a specific period.
2. Transaction-Based Pricing for Channelized WSs where customers are charged on the basis of transactions count over a specified period.
3. Risk-Based Model Pricing for Customized WSs where customers are charged on the basis of on-off payment methods for using the service over a specified period.

Tian et al. [27] categorized the provided services into three categories according to the WS QoS, namely Platinum, Gold, and Bronze. The price of 200 WS requests per second using the Platinum class is 0.05€. Similarly, the price of 150 requests per second using the Gold class is 0.03€. Likewise, the price of

100 requests per second using the Bronze class is 0.01€. Hence it is concluded that there is no clear pricing structure for using the WSs [25–28].

It is revealed from the previous discussion that there exists a room for applying the optimization techniques to the selection and construction procedures to minimize the number of integrated WSs for effective and efficient utilization of WS resources. In this research, we introduce an efficient technique to find the minimum number of services needed for selection and construction of services compositions in real large-scale WS repository. We also optimize the reusability and utilization of WS available resources.

3 The Proposed Work

In this study, we propose a technique that uses the SMFS to improve the WS selection and composition processes and optimize the utilization of the WS available resources. In this section, we discuss the Orchestrator modifications required to implement the proposed SMFS technique where the modifications are applied to the Business Process Execution Engine (BPEE) Orchestrator.

3.1 Modified BPEE Architecture

In this study, we have modified the architecture of BPEE to achieve enhancement in the execution process of the Orchestrator. Fig. 1 shows the modifications applied to the execution engine.

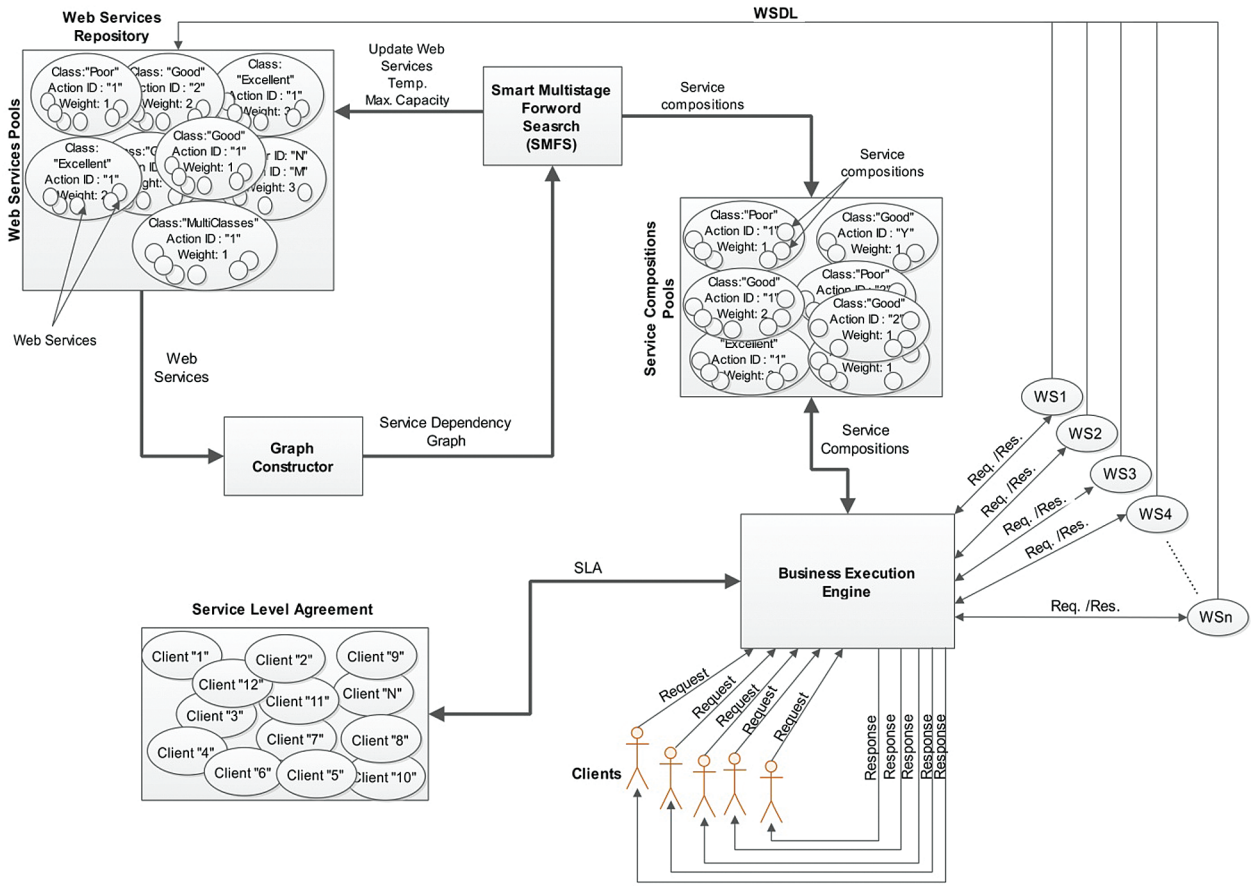

Figure 1: Modified BPEE architecture

Following is the list of modifications applied to the BPEE architecture.

1. Added a modified version of the WSs pool as suggested by Al-Hadid and Abu-Taieh [3] to classify the WSs as Excellent, Good or Poor according to the QoS attributes.
2. Added a novel WSs Multi-Class pool to handle the unused WSs as well as WSs with available resources that can be shared with different SCs.
3. Added the SMFS technique to enhance the WS selection as well as SC processes and deal with the unused WSs. SMFS filters and sends WSs having available resources to the Multi-Class pool of WSs after participating in SCs. The Multi-Class pool is used as a WS dataset repository pool to repeat the WS selection and SC processes.
4. Modified the WS pool architecture by adding the Temp. Max. Capacity (TempMaxCap) and the WS cost where TempMaxCap is the value of available WS resources that can be used after integrating the WSs in SC. The modified WS pool architecture is given in Tab. 1.

Table 1: WS repository structure

WS ID	Response Time	Availability	Weight	Action	WS Cost	Max. Capacity	TempMaxCap

Detailed description of different fields of the architecture is given below.

(1) **WS ID** is a unique identification number used as a reference to the WS in the execution engine.

(2) **Response Time** is the time needed to receive a response (ms) after sending a request. Response time is one of the important QoS attributes used to classify WSs. Tab. 2 shows the classification of WSs and SCs according to the response time suggested by [11,23,29].

Table 2: Classifications of WSs and SCs response time [23,29]

Class	WS	SC (five WSs)
Excellent response time	≥0.1–0.5 Sec	≤2.5 Sec
Good response time	>0.5–1.0 Sec	>2.5–5.0Sec
Poor response time	>1.0–3 Sec	>5.0–15 Sec

(3) **Availability** refers to the number of successful invocations/total invocations (percentage). The WS availability is one of the most important QoS attributes that is used to decide which class the WS or the SC belongs to. Following the idea of [11] and [29], Tab. 3 shows the WS and SC classifications according to their availability value.

Table 3: Classifications of the WS and SC availability [29]

Value	Class
[98–100%]	Excellent Availability
[96%–98%]	Good Availability
[90%–96%]	Poor Availability

(4) **WS cost**, mentioned in section 2, is not universally structured and there is no clear pricing method for the WSs [25–28]. Therefore, we use the QoS attributes in pricing the WS cost. Tab. 4 shows the cost for using the WSs according to the number of requests per second.

Table 4: WSs costs

Price per service usage	Throughput (Req. /Sec.)	Class
0.1$	600 Req/Sec	Excellent
0.05$	600 Req/Sec	Good
0.001$	600 Req/Sec	Poor

(5) **Weight** represents the classification of WSs on the basis of WS response time and availability. The weight of the WS can be calculated using Eq. 1 where the WS response time and availability will be mapped to represent 50% of weight each

$$\text{Weight (\%)} = \text{Response Time \%} + \text{Availability (\%)} \quad (1)$$

Using Eq. 1, WSs are classified into three classes as shown in Tab. 5. Note that, the classes are divided into intervals that cover all the ranges from 90% to 100% of the WS QoS and the SLA attributes.

Table 5: WSs classifications based on weight

Weight Value	Class
[98%–100%]	Excellent Class
[96% –98%]	Good Class
[90% –96%]	Poor Class

(6) **Action** represents the effect of a WS process. The Action value is used as a stage of service in the dependency graph G.

(7) **Max. Capacity** shows the maximum number of requests that can be accepted and processed by the WS per second. The Max. Capacity value ranges from (600–1800) requests per second.

(8) **TempMaxCap** is the value of available number of requests that can be accepted after the WS is engaged with SC(s).

Tab. 6 shows comparison of MFS and SMFS where MFS is used to enhance the WS composition and the SMFS is used to improve the WS selection and SC to optimize the WS resources utilization.

3.2 The Proposed SMFS Technique

The proposed SMFS technique has several phases. In the first phase, multistage dependency graph is constructed from appropriate WSs, which represent the graph nodes. The WSs are then divided into a set of stages according to the WS actions that represent the functionality and the output of the WSs. In the second phase, SMFS technique is applied to find the best SCs by optimizing the WS resource utilization. A formal description of the proposed SMFS technique is given below.

(1) Construct a multistage service dependency graph with all relevant WSs selected from an external repository according to each WS class pool.

Table 6: Comparison between MFS and SMFS

MFS	SMFS
All WSs are classified into pools based on the WS class (Excellent, Good, Poor) with all the WSDL Data.	In addition to the WS classification into pools (Excellent, Good, Poor), SMFS added new WS pool called Multi-Class to store all the unused WSs and the WSs having available resources after the composition process. Also, the TempMaxCap QoS attribute of the WS is added to all the pools where the TempMaxCap attribute is used to store the WS available resources after the SCs are constructed.
Each SC is created using WSs from the same pool; if the SC provides an Excellent Service this means that all the integrated WSs in the SC are selected from the Excellent pool.	In addition to the creation of SCs using WSs from the same pool, there are SCs that are constructed from different WS classes located in the Multi-Class pool. According to the availability of the WS resources, the WS can be used to construct new SCs.
The number of SCs depends on the available number of WSs in the pool related to the same class.	Besides, the SCs are constructed using WSs located in the same pool, an additional number of SCs can be constructed using the WSs located in the Multi-Class pool, which satisfy the minimum SLA requirements to construct the SC.
The SC maximum capacity is the maximum number of requests that can be accepted by the SC and it depends on the WS maximum capacity integrated in the SC. Here, the SC maximum capacity relies on the minimum value of the WS maximum capacity in the SC.	Even though, the SC maximum capacity depends on the minimum value of the WS maximum capacity in the SC, the proposed algorithm uses the WS remaining resources and the unused WSs to create new SCs and increase the system's capacity to handle more requests. The process is applied after the construction of SCs where the unused WSs and the WSs having available resources are utilized to create new SCs from different WS classes located in the Multi-Class pool, which are used to satisfy the clients' demands and the SLA requirements.

Graph: Given a set of WSs named as WS, SC is defined as a tuple $SC = \{In_{ws}, Out_{ws}\}$ where $In_{ws} = \{In_{ws}^{1}, In_{ws}^{2}\ldots, In_{ws}^{n}\}$ is a set of inputs and $Out_{ws} = \{Out_{ws}^{1}, Out_{ws}^{2}\ldots, Out_{ws}^{n}\}$ is a set of outputs. Here, In_{ws} is required to invoke the WS and generate the set of WSs output Out_{ws}. Both the In_{ws} and Out_{ws} are used to combine the matched inputs and outputs of the WS to construct the SC.

(2) Divide the WS into sub-sets that represent the stages based on the roles described in the WSDL and the output of each WS.

$S = \{w_1, w_2, w_3, \ldots..,w_n\}$

$S = \{ \{ w_1, \{w_2, w_3 \}, w_4\}, \{\{w_5, w_6\}, w_{7,} w_8\}, \{w_9, w_{10,} w_{11}\}\ldots \{w_{n-2}, w_{n-1}, w_n \}\}$

where $\{w_2, w_3\}$ and $\{w_5, w_6\}$ are parallel WSs, which achieve the task together just like a single WS such as w_7 and w_9. Fig. 2 illustrates the WS Graph G.

```
// number of stages in the graph G
1; // stages loop
// number of tuples in graph G
        Do until( for all S ∈ S_R)
Do until ( i = k)
        Set all stages values in the tuple to INF;
Loop // stages loop
For each S ∈ S_R
        If S ∉ G & S_Action = k & S_k ∈ S_k-1(L) & S_TempMaxCap > 0 then
                Add S to graph G
++T; // next tuple in the graph G
Loop // tuples loop
INF: infinity
S ∈ S_R: S is the WS and SR is the WS repository)
S_k ∈ S_k-1(L): There are links to the S from the prior S
```

Figure 2: WSs DGCA

Tuple: Graph elements are grouped according to the WSs' actions where similar WSs are clustered in the same stage of the WS graph, which is represented as a matrix. Each tuple in the matrix represents the edges to the next WS based on the tuple order in the matrix. The values in the tuple are the costs to invoke the WS. The position of the WS in the tuple corresponds to the WS order in the graph. If there is an infinity value (INF) in the tuple, this means that there is no edge to the corresponding WS, which can be derived from WS order in the tuple.

Tuple = $\{WS1_{Cost}, WS2_{Cost}, WS3_{Cost}, WS4_{Cost} \ldots WSN_{Cost}\}$

Fig. 2 shows the WSs Dependency Graph's Construction Algorithm (DGCA), which generates the WSs Graph (G) as shown in Fig. 3.

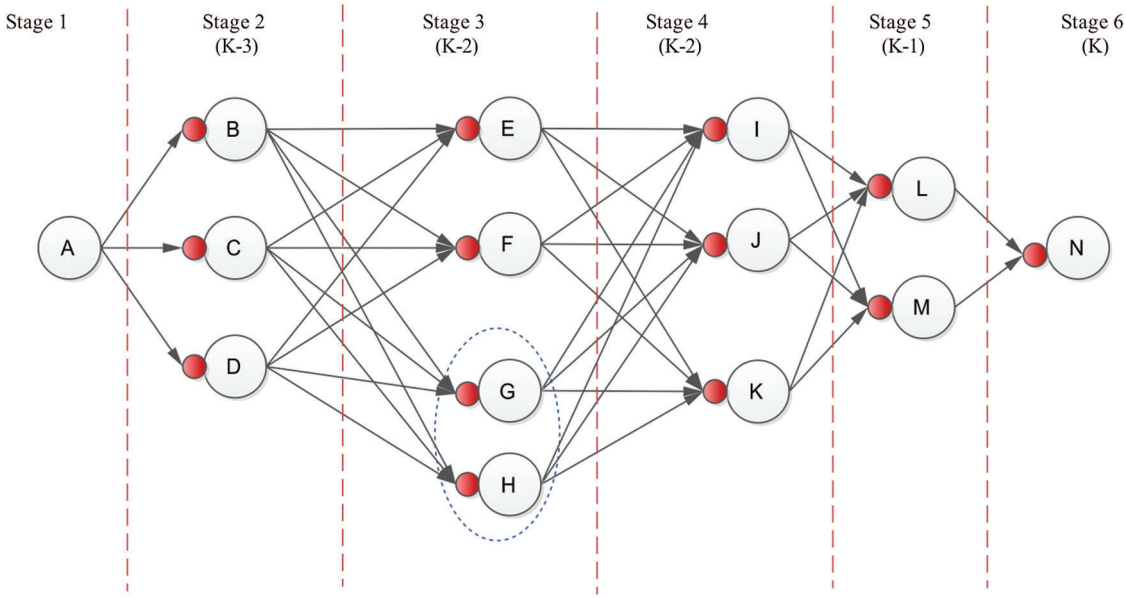

Figure 3: Constructed WSs graph (G)

(3) For each WS in the constructed graph G, set the TempMaxCap = MaxCap.

(4) Minimize the SC using MFS based on cost of each WS and then find the minimum cost of the composite WS that forms the SC.

Tuple = {WSA_{Cost}, WSB_{Cost}, WSC_{Cost}, WSD_{Cost}, WSE_{Cost}, WSF$_{Cost}$, WSG_{Cost}, WSH_{Cost}, WSI_{Cost}, WSJ_{Cost}, WSK_{Cost}, WSL_{Cost}, WSM_{Cost}, WSN_{Cost}}

Fig. 4 illustrates the repository matrix that represents the available WSs and the corresponding cost of each WS.

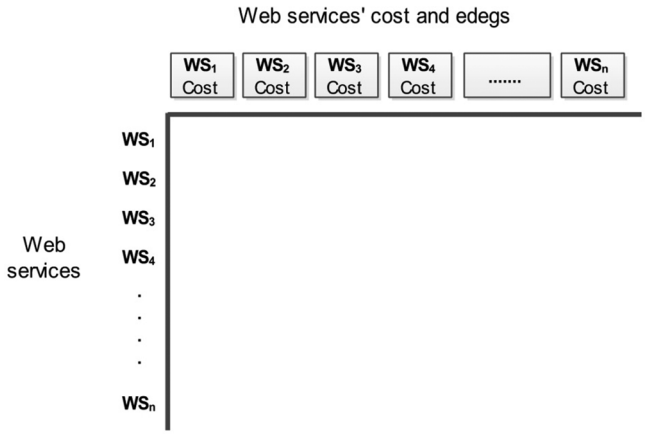

Figure 4: WSs repository matrix

The WSs from Fig. 3 are represented as a set G matrix that contains all the WSs in the repository. Each tuple in the G matrix is composed of a source WS and a set of WSs that are reachable from that source. For example, the first tuple shows that WSB, WSC and WSD are reachable from WSA on the corresponding cost of each invocation. Meanwhile, the other WSs with *INF* costs are not reachable and cannot be invoked by WSA.

G = { WS*A:* {INF, WSB_{Cost}, WSC_{Cost}, WSD_{Cost}, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF}, WS*B:* {INF, INF, INF, INF, WSE_{Cost}, WSF$_{Cost}$, WSG_{Cost}, WSH_{Cost}, INF, INF, INF, INF, INF, INF}, WS*C:* { INF, INF, INF, INF, WSE_{Cost}, WSF$_{Cost}$, WSG_{Cost}, WSH_{Cost}, INF, INF, INF, INF, INF, INF}, WS*D:* { INF, INF, INF, INF, WSE_{Cost}, WSF$_{Cost}$, WSG_{Cost}, WSH_{Cost}, INF, INF, INF, INF, INF, INF}, WS*E:* { INF, INF, INF, INF, INF, INF, INF, INF, WSI_{Cost}, WSj_{Cost}, WSk_{Cost}, INF, INF, INF}, WS*F:* { INF, INF, INF, INF, INF, INF, INF, INF, WSI_{Cost}, WSj_{Cost}, WSk_{Cost}, INF, INF, INF}, WS*G:* { INF, INF, INF, INF, INF, INF, INF, INF, WSI_{Cost}, WSj_{Cost}, WSk_{Cost}, INF, INF, INF}, WS*H:* { INF, INF, INF, INF, INF, INF, INF, INF, WSI_{Cost}, WSj_{Cost}, WSk_{Cost}, INF, INF, INF}, WS*I:* { INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, WSL_{Cost}, WSM_{Cost}, INF}, WS*J:* { INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, WSL_{Cost}, WSM_{Cost}, INF}, WS*K:* { INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, WSL_{Cost}, WSM_{Cost}, INF}, WS*L:* { INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, WSN_{Cost}}, WS*M:* { INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, WSN_{Cost}}}

(5) Enhance the service selection and composition using the proposed SMFS technique and optimize the utilization of the WSs available resources. Fig. 5 shows the proposed SMFS technique.

(6) Reconstruct the multistage service dependency graph with all relevant WSs elected from an external repository (Excellent, Good, and Poor WSs pools) after the composition process and update the WS TempMaxCap using Eq. 3, then go back to step (3).

```
var K; // number of stages in the graph G
var i = K -2; // current stage
var L; // all edges connected to j in the next stage (i+1)
Do until ( No More SC can be constructed )
Do until ( i = 1 )
Cost(i, j) = Min {C(i, L) + cost (i+1) , L)} where L in V_{i+1} & (j, L) ∈ G.        (2)
--i; // prior stage
Loop // stages loop
//after the SC is constructed
      for each WS ∈ constructed service composition
        if TempMaxCap > min. value of the TempMaxCap ϵ SC
        TempMaxCap = MaxCap -  Min_Value(MaxCap)                                    (3)
        Else if TempMaxCap = min. value of the TempMaxCap ϵ SC
 TempMaxCap = INF.....
Loop // Find next SC
 Do Until (No More WSs in the graph G)
if Web service TempMaxCap ≠ INF
// WSs with available resources can be used to construct new service compositions.
Copy WS to Multi-Class Pool in the WS Repository. Next WS
```

Figure 5: The proposed SMFS technique

In SMFS Algorithm, Eq. 2 is used to find the WS minimum cost that meets the SLA requirements for each stage starting from Cost (k-2, j) for all j in V_{k-2} until we reach the Cost (1, S), for all j in V_1 which is the WS execution engine considered as the source of the invocations. Eq. 3 is used to update TempMaxCap for all WSs by subtracting the minimum value of the maximum capacity. This is related to the WSs, which are used in the construction of SC. Involving the maximum capacity of each WS in the calculations makes sure that we are utilizing the available resources optimally.

After finding the minimum cost of the composition service and updating the WS TempMaxCap, all WSs with available resources (TempMaxCap > 0) are located in the Multi-Class pool. The WSs in the Multi-Class pool are used to create new web compositions and provide services to clients that might be related to same or different classes. Newly created SCs are classified using Eq. 1 to decide which SC class it belongs to.

4 Experimental Results and Discussion

4.1 Simulation Setup

The proposed algorithm is implemented using the Hybrid Service Oriented Architecture Simulator (HSOAS) [30] for WS selection and composition. The simulator supports the service classification and composition in addition to the SLA Gap, which is used to demonstrate the behavior of WSs and QoS fluctuations.

All simulation experiments were performed on i7-3632QM machine equipped with 2.20 GHz processor and 8 GB DDR3 RAM. Tab. 7 shows the initial values of the simulator parameters.

4.2 Datasets

In order to evaluate the proposed service selection and composition, the simulator is used to build the WSs repository using the QWS dataset proposed by Al-Masri and Mahmoud [31]. This dataset is a labeled dataset describing real-world QoS evaluations with response time and availability as parameters that are compiled from the results of 2509 WSs.

In addition to the available ranked WSs according to the QoS attributes in QWS dataset, HSOAS added new attributes to WS repository needed by classification and composition processes for the implementation of proposed technique. The attributes are Action, Max. Capacity and TempMaxCap as discussed in section 3.1 (Tab. 1).

Table 7: HSOAS initialization values of the parameters

Parameter	Value
Number of runs	20
Number of clients/run	1000
Min. Number of requests/Client	50
Max. Number of requests/Client	5000
Number of WSs	2509
Number of Clients	200
Min Value of WSs maximum capacity level	600
Max Value of WSs maximum capacity level	1800

4.3 Experimental Results and Discussion

In this section, we assess the performance of our proposed SMFS technique using the QWS dataset. SMFS technique is designed to optimize the WS selection and construction processes. The performance of SMFS and MFS techniques are compared for in depth analysis. The performance of SMFS technique is also evaluated for resourceful utilization of WS remaining resources and the unused WSs after the SC process is completed in the first round.

Fig. 6 depicts the average number of WSs categorized in each class pool where the total number of WSs in the QWS dataset is 2508. The WSs in Excellent, Good, and Poor classes are 830, 831, and 847, respectively.

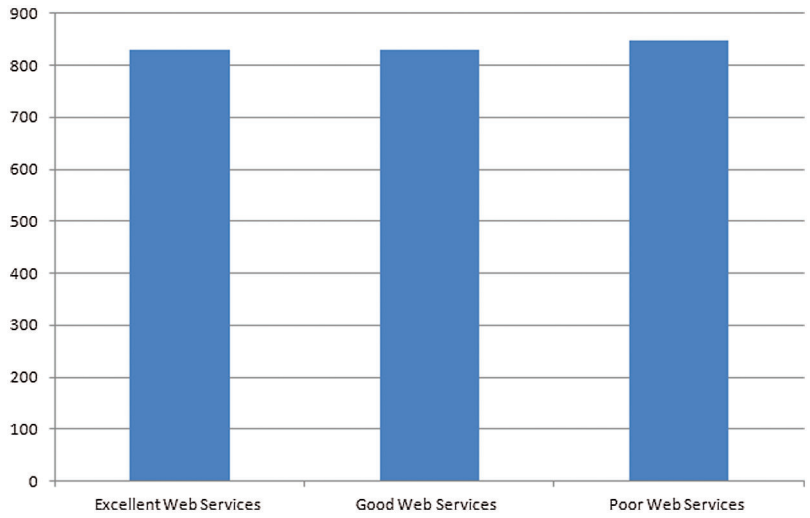

Figure 6: Average number of WSs in each WS pool class: Excellent, Good, and Poor

Fig. 7 shows the comparative performance of MFS and SMFS techniques in terms of WS resources utilization. The proposed SMFS technique demonstrated higher performance compared to the MFS technique as evident from the obtained results. The enhanced performance of SMFS technique is due to its capability of using the remaining WS resources in the construction of new SCs after the first round of SC process is over. The WS resources are not utilized completely with MFS technique due to the

maximum capacity of integrated WSs that exceed the maximum capacity of SC. The performance of SMFS technique in terms of WS resource utilization is enhanced by 9.6% compared to the standard MFS.

Figure 7: The number of WSs constructed by MFS and SMFS techniques as Excellent, Good, and Poor

Fig. 8 illustrates the left-over WSs of SMFS and MFS techniques. In case of SMFS technique, the number of unused WSs are decreased, which shows that the proposed SMFS technique achieved maximum utilization of WSs. This indicates that the number of constructed SCs with SMFS technique is increased by entertaining more number of requests.

Figure 8: The number of left-over WSs in the SCs for the Excellent, Good, and Poor classes using MFS and SMFS techniques

The average number of constructed SCs and the WS consumed resources using MFS and SMFS techniques is shown in Figs. 9 and 10.

As evident from Fig. 9, the average number of constructed SCs using SMFS technique exceeds the average number of constructed SCs using MFS technique. The SMFS technique constructed 36.97% more SCs compared to MFS technique. The SMFS technique constructed additional SCs from the WSs

stored in the Multi-Class pool after the first round of composition process is over. This shows the effectiveness of Multi-Class pool in constructing additional SCs.

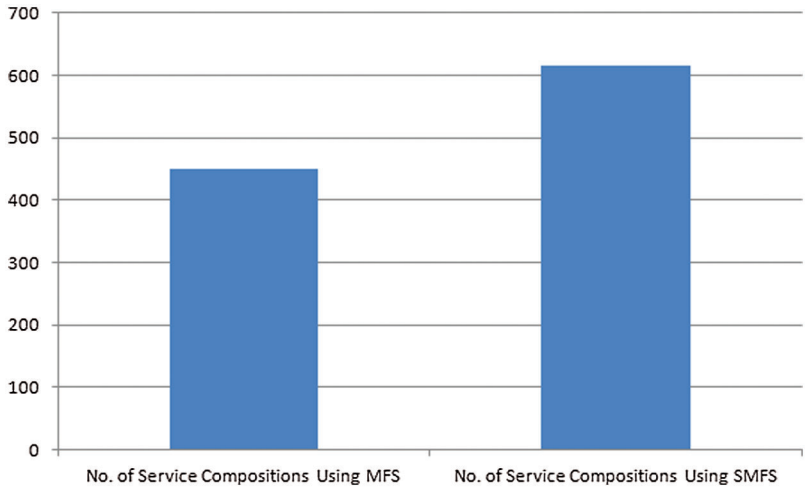

Figure 9: The average number of SCs using MFS and SMFS techniques for all WS classes (Excellent, Good, and Poor)

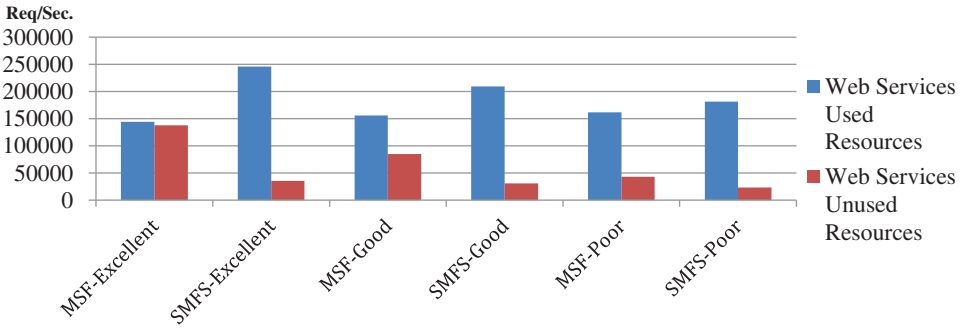

Figure 10: Average number of used/unused resource after the SC process using MFS and SMFS for all WS classes (Excellent, Good, and Poor)

Fig. 10 highlights the average number of used and unused WS resources using the MFS and SMFS techniques after the SC process is over. The proposed SMFS technique maximized the utilization of WS resources and hence the left-over WS resources are minimized. The demonstrated performance of SMFS technique is better than that of MFS technique. The outstanding performance of SMFS technique is due to its capability of utilizing the WSs located in the Multi-Class pool containing integrated and unintegrated WSs. The SMFS employs those WSs from the Multi-Class pool to construct new SCs and the hence optimizes the WS resource utilization.

The previous discussion confirms that SMFS technique for WS selection and SC improves the WS resource utilization and increases the number of constructed SCs. As a result, it delivers enhanced services to the clients and guarantees the fulfillment of SLA requirements.

The optimal utilization of WS resources is due to the fact that SMFS technique constructs additional SCs using the unused WSs from the Multi-Class pool after the first round of SC process. Additionally, it uses the

WSs that are already integrated in SCs but still have available resources according to the value of the maximum capacity as discussed in section 3.2.

Though the performance of our proposed SMFS technique is outstanding, it comes at the cost of additional computational time. The time needed by the SMFS process to collect and construct new SCs is more than the time required by the MFS technique. The following equations are used to find the time complexity of the algorithms where Eq. 4 is used to calculate the time complexity of the MFS algorithm and Eq. 5 is used to determine the time complexity of SMFS technique.

$$T(n) = O(\boldsymbol{\alpha}^2) + O(\boldsymbol{\beta}^2) + O(\boldsymbol{\gamma}^2) \tag{4}$$

$$T(n) = O(\boldsymbol{\alpha}^2) + O(\boldsymbol{\beta}^2) + O(\boldsymbol{\gamma}^2) + O(\boldsymbol{x}^2) \tag{5}$$

where $\boldsymbol{n}$ is the number of WSs in the graph $\boldsymbol{G}$, $\boldsymbol{\alpha}$ is the total number of WSs in the "Excellent pool," $\boldsymbol{\beta}$ is the WSs in the "Good pool," $\boldsymbol{\gamma}$ is the WSs in the "Poor pool," $\boldsymbol{x}$ is the number of WSs that are located in the Multi-Class pool.

Fig. 11 shows that the time needed to construct the SCs using SMFS technique is more than the time required to construct the SCs using MFS technique where the extra time $O(x^2)$ is needed by SMFS to search in the Multi-Class pool and construct new SCs.

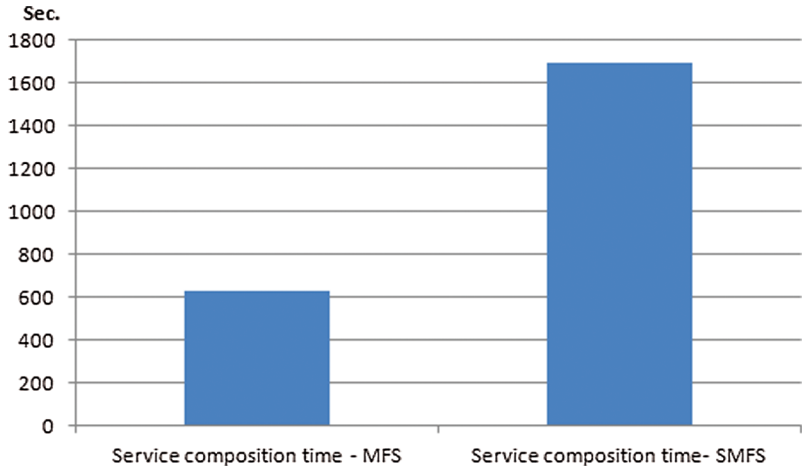

Figure 11: Time required to construct SCs using MFS and SMFS techniques

5 Conclusion

SOA is used to connect different applications and technologies using the WS components where the WS offers simple or complex tasks by SCs. One of the main challenges of the SCs construction process is selecting the right WS to provide the expected QoS to the clients according to the SLA requirements.

This research presents SMFS based effective technique to optimize the WS resource utilization and improves the WS selection and SC by improving the Orchestrator processes. The proposed technique can also create new SCs using the unused and integrated WSs located in the Multi-Class pool. The proposed SMFS technique improved the WS resource utilization by 9.6% and increased the number of constructed SCs by 36.97% in comparison with the MFS technique. It can be noted that the enhanced performance of SMFS technique is on the cost of additional computational time that is required to collect, select, and construct new SCs. The additional tasks associated with SMFS technique include updating the value of

available resources for each WS, creating the new graph using the WS having available resources collected from different pools in Multi-Class pool, and creating new SCs.

As a future endeavor, it would be valuable to enhance the proposed SMFS technique by minimizing the computational time. Moreover, in some cases a specific resource may not be available or missing due to different reasons for example, network failure, server maintenance, buffer size, etc. Therefore, adding a process that allows the SMFS technique to discover and integrate new WSs to compensate the unreachable or unavailable WSs and satisfy the SCs QoS requirements.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflict of interest to report regarding the present study.

References

[1] K. Skałkowski and K. Zieliński, "Automatic adaptation of SoA systems supported by machine learning," in *Doctoral Conf. on Computing, Electrical and Industrial Systems*, pp. 61–68, 2013.

[2] S. Afaneh and I. Al Hadid, "Airport enterprise service bus with three levels self-healing architecture (AESB-3LSH)," *Int. Journal of Space Technology Management and Innovation (IJSTMI)*, vol. 3, no. 2, pp. 1–23, 2013.

[3] I. Al-Hadid and E. Abu-Taieh, "Web services composition using dynamic classification and simulated annealing," *Modern Applied Science*, vol. 12, no. 11, pp. 376–386, 2018.

[4] A. H. Issam, "Airport enterprise service bus with self-healing architecture (aesb-sh)," *Int. Journal of Aviation Technology, Engineering and Management (IJATEM)*, vol. 1, no. 1, pp. 1–13, 2011.

[5] I. Sommerville, *Introduction to Software Engineering*. 10th. ed. Boston, MA: Addison-Wesley, 2007.

[6] R. Karunamurthy, F. Khendek and R. H. Glitho, "A novel architecture for Web service composition," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 787–802, 2012.

[7] S. Khwaldeh, E. Abu-Taieh, I. Alhadid, R. Alkhawaldeh and R. E. Masa'deh, "Dyorch: dynamic orchestrator for Improving web services composition," in *Int. Business Information Management Conf. (33rd IBIMA)*, Granada, Spain, 2019.

[8] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam and Q. Z. Sheng, "Quality driven web services composition," in *Proc. of the 12th Int. Conf. on World Wide Web*, pp. 411–421, 2003.

[9] M. Karimi, F. S. Esfahani and N. Noorafza, "Improving response time of web service composition based on QoS properties," *Indian Journal of Science and Technology*, vol. 8, no. 16, pp. 1–4, 2015.

[10] G. N. Rai, G. Gangadharan, V. Padmanabhan and R. Buyya, "Web service interaction modeling and verification using recursive composition algebra," *IEEE Trans. on Services Computing*, vol. 14, pp. 300–314, 2021.

[11] C. Jatoth, G. Gangadharan, U. Fiore and R. J. Buyya, "Qos-aware Big service composition using MapReduce based evolutionary algorithm with guided mutation," *Future Generation Computer Systems*, vol. 86, no. 7, pp. 1008–1018, 2018.

[12] C. Wang, H. Ma, G. Chen and S. Hartmann, "A memetic NSGA-II with EDA-based local search for fully automated multiobjective web service composition," in *Proc. of the Genetic and Evolutionary Computation Conf*, Companion, pp. 421–422, 2019.

[13] M. B. Juric, B. Mathew and P. G. Sarang, *Business Process Execution Language for Web Services: An Architect and Developer's Guide to Orchestrating Web Services using BPEL4WS*. Birmingham, United Kingdom: Packt, 2006.

[14] V. Muthusamy, H.-A. Jacobsen, T. Chau, A. Chan and P. Coulthard, "Sla-driven business process management in soa," in *Proc. of the 2009 Conf. of the Center for Advanced Studies on Collaborative Research*, USA, pp. 86–100, 2009.

[15] S.-L. Fan, Y.-B. Yang and X.-X. Wang, "Efficient web service composition via knapsack-variant algorithm," *Int. Conf. on Services Computing*, vol. 10969, pp. 51–66, 2018.

[16] S. Mirzayi and V. Rafe, "A hybrid heuristic workflow scheduling algorithm for cloud computing environments," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 27, no. 6, pp. 721–735, 2015.

[17] J. Jung, B. Krishnamurthy and M. Rabinovich, "Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites," in *Proc. of the 11th Int. Conf. on World Wide Web*, pp. 293–304, 2002.

[18] D. K. Yau, J. C. Lui, F. Liang and Y. Yam, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," *IEEE/ACM Trans. on Networking*, vol. 13, no. 1, pp. 29–42, 2005 2005.

[19] Y. Gao, J. Na, B. Zhang, L. Yang and Q. Gong, "Optimal web services selection using dynamic programming," in *11th IEEE Symp. on Computers and Communications (ISCC'06)*, pp. 365–370, 2006.

[20] S. L. Tilahun, J. M. T. Ngnotchouye and N. N. Hamadneh, "Continuous versions of firefly algorithm: a review," *Artificial Intelligence Review*, vol. 51, no. 3, pp. 445–492, 2019.

[21] S. Udhaya Shree, A. Amuthan and K. Suresh Joseph, "Integrated ant colony and artificial bee colony optimization metaheuristic mechanism for quality of service based web service composition," *Journal of Computational and Theoretical Nanoscience*, vol. 16, no. 4, pp. 1444–1453, 2019.

[22] J.-Y. Jung, J. Bae and L. Liu, "Hierarchical clustering of business process models," *Int. Journal of Innovative Computing, Information and Control*, vol. 5, no. 12, pp. 1349–4198, 2009.

[23] Z.-p Gao, C. Jian, X.-s Qiu and L.-m Meng, "Qoe/qos driven simulated annealing-based genetic algorithm for Web services selection," *Journal of China Universities of Posts and Telecommunications*, vol. 16, no. 6, pp. 102–107, 2009.

[24] H. Elmaghraoui, I. Zaoui, D. Chiadmi and L. Benhlima, "Graph based e-government web service composition. *arXiv preprint arXiv,* 2011.

[25] Z. Lin, H. Zhao and S. Ramanathan, "Pricing web services for optimizing resource allocation an implementation scheme," in *Proc. Web2003*, Seattle, WA, USA, pp. 1–7, 2003.

[26] G. E. Mathew, J. Shields and V. Verma, "Qos based pricing for web services," *Int. Conf. on Web Information Systems Engineering*, vol. 3307, pp. 264–275, 2004.

[27] M. Tian, A. Gramm, T. Naumowicz, H. Ritter and J. Freie, "A concept for qos integration in web services," in *Fourth Proc. Int. Conf. on Web Information Systems Engineering Workshops*, pp. 149–155, 2003.

[28] D. A. Menasce, "Composing web services: a QoS view," *IEEE Internet Computing*, vol. 8, no. 6, pp. 88–90, 2004.

[29] H. Ludwig, A. Keller, A. Dan, R. P. King and R. Franck, *Web service level agreement (WSLA) language specification*, New York, United States: IBM Corporation, 2003.

[30] H. Al-Tarawneh, I. AlHadid, K. kaabneh and A. Alhroob, "Hybrid service oriented architecture simulator," in *3rd Int. Computer Sciences and Informatics Conf. (ICSIC 2019)*, 2019.

[31] E. Al-Masri and Q. Mahmoud, "Toward quality-driven web service discovery," *IT Professional*, vol. 10, no. 3, pp. 24–28, 2008.