symmetry

MDPI

Article

An Intelligent Cloud Service Composition Optimization Using Spider Monkey and Multistage Forward Search Algorithms

Hassan Tarawneh [1], Issam Alhadid [2], Sufian Khwaldeh [2,*] and Suha Afaneh [3]

[1] Faculty of Information Technology, Al-Ahliyya Amman University, Amman 19328, Jordan; h.altarawneh@ammanu.edu.jo
[2] Faculty of Information Technology and Systems, University of Jordan, Aqaba 77111, Jordan; i.alhadid@ju.edu.jo
[3] Society of Arabic Language Computerization and Enrichment of Arabic E-Content, Amman 11143, Jordan; suha.afaneh1211@gmail.com
* Correspondence: skhwaldeh@googlemail.com

Abstract: Web service composition allows developers to create and deploy applications that take advantage of the capabilities of service-oriented computing. Such applications provide the developers with reusability opportunities as well as seamless access to a wide range of services that provide simple and complex tasks to meet the clients' requests in accordance with the service-level agreement (SLA) requirements. Web service composition issues have been addressed as a significant area of research to select the right web services that provide the expected quality of service (QoS) and attain the clients' SLA. The proposed model enhances the processes of web service selection and composition by minimizing the number of integrated Web Services, using the Multistage Forward Search (MSF). In addition, the proposed model uses the Spider Monkey Optimization (SMO) algorithm, which improves the services provided with regards to fundamentals of service composition methods symmetry and variations. It achieves that by minimizing the response time of the service compositions by employing the Load Balancer to distribute the workload. It finds the right balance between the Virtual Machines (VM) resources, processing capacity, and the services composition capabilities. Furthermore, it enhances the resource utilization of Web Services and optimizes the resources' reusability effectively and efficiently. The experimental results will be compared with the composition results of the Smart Multistage Forward Search (SMFS) technique to prove the superiority, robustness, and effectiveness of the proposed model. The experimental results show that the proposed SMO model decreases the service composition construction time by 40.4%, compared to the composition time required by the SMFS technique. The experimental results also show that SMO increases the number of integrated ted web services in the service composition by 11.7%, in comparison with the results of the SMFS technique. In addition, the dynamic behavior of the SMO improves the proposed model's throughput where the average number of the requests that the service compositions processed successfully increased by 1.25% compared to the throughput of the SMFS technique. Furthermore, the proposed model decreases the service compositions' response time by 0.25 s, 0.69 s, and 5.35 s for the Excellent, Good, and Poor classes respectively compared to the results of the SMFS Service composition response times related to the same classes.

Keywords: web service; composition; optimization; Spider Monkey; Multistage Forward Search

Citation: Tarawneh, H.; Alhadid, I.; Khwaldeh, S.; Afaneh, S. An Intelligent Cloud Service Composition Optimization Using Spider Monkey and Multistage Forward Search Algorithms. *Symmetry* **2022**, *14*, 82. https://doi.org/10.3390/sym14010082

Academic Editor: Deming Lei

Received: 6 November 2021
Accepted: 7 December 2021
Published: 5 January 2022

1. Introduction

Service-Oriented Architecture (SOA) is a style of software design that utilizes Web Services. SOA creates applications and services using different technologies. SOA has been adopted by organizations to implement simple or complex tasks and provide business processes workflow called service composition.

Service composition is a workflow organized and supervised by the Service Execution Engine. Each service composition consists of several web services that can be managed,

replaced, and updated at runtime, without interrupting the ongoing business processes that affect the Quality of Service (QoS) attributes [1–4].

The QoS is related to the integrated web services that compose the service composition, which might be affected by different internal or external factors such as hosting environment, network, and service upgrade [2,4,5], the problems of web service scalability, performance, and the ability to respond to a massive number of synchronous requests while satisfying the expected SLA requirements, functionality, and behavior [6–9].

Selecting the right web service directly affects the constructed service composition QoS. There are two types of methods and techniques that are used to select and construct the service compositions: static and dynamic methods. Static methods and techniques are based on the idea of selecting the available web services to construct the service compositions that achieve the required task. This type is usually used when the number of web services is limited, and the number of synchronous requests can be handled by constructed service compositions [10,11]. The dynamic methods and techniques are used when there is a massive number of web services that can be used to construct a substantial number of service compositions. The integrated web services can be replaced automatically to provide and satisfy the clients' expected SLA requirements [12,13]. Dynamic methods and techniques are complex processes that satisfy the SLA functional and non-functional requirements. These requirements' constraints are still a challenge, and they are not efficient enough when applied in real-time and large-scale environments, in addition to the mentioned factors that affect the QoS of the web services [8,14,15]. It is still a complex process to find web services that provide identical functionality, with the required resources to create the service composition on a large-scale web services repository [16]. Uc-Cetina et al. [6] claimed that web services selection and composition processes complexity involve three main factors: the available substantial number of web services with similar functionality, the different possibilities of integrating web services into a service composition, and various QoS requirements of the service composition.

Other problems that may affect the service composition QoS include the utilization of web services' resources. This issue relates to the value of the maximum capacity of integrated web services. Maximum capacity is the maximum number of requests that can be accepted and handled by the service, per second. The maximum capacity of the integrated web services is determined by the maximum capacity of the web service with the least value [8]. This means that the other web services (web services excluding the one with the least value) will not have their leftover requests utilized, even when the clients request the services at peak times. Moreover, this research has considered the framework proposed by Dubey et al. [17]. This framework proposed the fundamentals of service composition methods symmetry and variations.

According to the discussed points, there are three major research perspectives to be analyzed:

1. Design a dynamic approach to improve the processes of the Orchestrator web service selection and service composition construction, to satisfy the set of predefined QoS constraints specified in the clients' SLA.
2. Control the workload on the web services and service compositions and upgrade the service composition dynamically. The following is achieved by replacing one or more of the integrated web services. This provides the identical functionality and QoS of the web service that is unavailable or with an unacceptable QoS.
3. Improve the service composition flexibility and scalability and optimize the integrated web services' resources utilization.

To achieve our goals, we propose a novel model based on the Spider Monkey Optimization (SMO) algorithm and adopt cloud computing capabilities.

Cloud computing is used to provide scalable and elastic service compositions using virtualization technology [18–20]. Virtualization technology enables resource sharing and the creation of multiple network virtual environments or resources, from a single processing machine [21,22]. In addition to the auto-scaling capabilities (to upgrade or downgrade the

server of the Virtual Machine (VM) capabilities), it helps meet the demand for resources by adding or removing VM capacity, which includes computing, storage, or network services; to maintain the services and SLA requirements [23]. Virtualization and auto-scaling provide the new VM with the same computational capabilities. This means that the execution will be in parallel, and the tasks will be divided on the VMs using the time sharing and memory sharing scheduler [18,21–23].

Virtualization uses a software layer called a hypervisor to coordinate between VM operating system and applications and the underlying physical hardware, where hypervisor separates the operating system and applications from the physical computer hardware, in addition, it allocates physical computing resources including processors, memory, and storage. Accordingly, new VMs are created on-demand to handle new tasks when the existing VM becomes over-utilized [21,23]. While the Cloud Load balancer is used to find the right balance between the VM resources and the service compositions capabilities. Where the VM resources include the processing Capacity, memory load, Computation (CPU) load, and network load. While the service compositions' capabilities related to the integrated web services resources, include handling, processing, and responding to the invocations.

Meanwhile, the SMO algorithm simulates the fission–fusion social structure of Spider Monkeys during their foraging behavior. SMO illustrates two essential concepts of swarm intelligence; self-organization and division of monkey groups (where SMO split or combine based on food scarcity or availability) [24]. The VM resources and processing capacity represent the food, while the web services represent the Spider Monkeys. When the workload on the VM resources is high, the VM utilizes the auto-scaling to split the VM into two or more VMs to increase the capabilities of the VMs. On the other hand, if the workload is high on the integrated web services group related to the service compositions, the web services' group splits into two or more groups. It does that to reduce the workload, handle more clients' requests, and provide the mandatory service within the SLA requirements. The SMO is intended to optimize the selection of service composition orchestration and upgrade the service composition and the service QoS.

VM resources and processing capacity are affected directly by the clients' requests; if the clients' requests increase, then the available VM resources will decrease. In that case, it may not be able to handle the requests and invoke the involved integrated web services of the service composition successfully.

When put together, SMO and cloud computing provide the capabilities to enhance the VMs and the web services' performance and resources by utilizing the Cloud Computing virtualization and the auto scaling capabilities.

To achieve our goals using the proposed model, classification on two distinct levels should be performed: classifying the web services according to the attributes of the QoS and grouping the web services into different classes according to their actions. Then using the SMO, identify a local leader for each class of the web service. The local leader is a Load Balancer VM that will monitor the service compositions in the class. After identifying the local leader; a dependency graph is constructed by the Multistage Forward Search (MFS) algorithm, which is used to find the best service compositions in the graph.

During the process of providing the services to the clients, the Load Balancer will control and monitor the web service, compositions workload, QoS, and capacity. Accordingly, the web services groups are updated either by combining or dividing the group and using the auto-scaling capabilities to upgrade or downgrade the VM capabilities. After the upgrade process, the dependency graph is either updated or recreated. The MFS algorithm will be used again to find the new service compositions in the updated graph and provide the service again to the clients.

This research presents the proposed model (using the SMO algorithm) to solve the service composition construction process and the web service resources utilization problems efficiently. The main contributions are:

- Using dependency graphs and the MSF algorithm to find the paths and the candidate service compositions in each group.
- Utilizing the SMO algorithm. Each web services class will be grouped according to the web services' QoS attributes.
- Assigning a VM and identifying a load balancer for each group, to control and monitor the workload on the provided services.
- Optimizing the utilization of the available web service resources so that they can be used in other service compositions.
- Tracking the service composition's QoS and accordingly, splitting or combining the web service groups using the VM virtualization and auto-scale.

This research is organized in the following manner: Section 2 introduces related literature about web services' selection and composition. In Section 3, the paper describes the proposed method, which includes modifications to the Service execution engine architecture and SMO algorithm. Section 4 discusses the simulation, dataset, results, and evaluation. Section 5 finally gives the research conclusions and the future directions.

2. Related Work

WS selection and composition have been addressed as one of the active research areas. This research aims to improve the WS performance and reliability, using different heuristic and non-heuristic approaches, to achieve the SLA requirements [16,25–27]. To achieve SLA requirements, some researchers suggest ignoring the clients' requests once the SLA's maximum capacity is exceeded. Yau et al. [28] proposed a method to avoid the denial of service and distributed denial of service attacks. while [5] recommended the prioritization of the clients' requests based on the SLA contract. In this connection, [8] proposed a dynamic mechanism to enhance the web service's selection and service composition processes. The mechanism was based on Simulated Annealing (SA) to achieve the SLA requirements, as well as improve the service compositions' availability and response time. On the other hand, several researchers employed metaheuristic optimization algorithms to solve the services' composition challenge [29–31]. In this regard, Shree et al. [30] proposed a method by combining the Ant Colony Optimization Algorithm with Artificial Bee Colony Optimization Algorithm (IACO-ABCOA), to find the optimal web service configuration solution and to solve the stagnation, as well as convergence problems. Jung et al. [32] proposed the cosine similarity-based method for business process clustering by identifying similar processes. The following was achieved by comparing and reengineering business process models to support new business process designs. Gao et al. [33] proposed an algorithm based on SA and Genetic Algorithm to optimize the process of web service selection and composition. Furthermore, Sangaiah et al. [34] proposed a method based on evolutionary optimization, using biogeography-based optimization (BBO).

Androcec et al. [35] addressed the problem of platform as a service interoperability. This issue appears when different API's from different vendors are delt with as service and as response, a new algorithm for identifying the interoperability problem alongside providing a specific application domain which is composed of operations defined in PaaS API's.

Mousaa and Bentahara [36] proposed a mathematical model to optimize the web services selection and composition using the Social Spider Algorithm (SSA). The SSA proposed model showed promising results. Where SSA overpassed the Particle Swarm Optimization (PSO) web services selection and composition results, in terms of both execution time and fitness, with different tasks and substitutable web services.

Elmaghraoui et al. [37] presented a method that is based on modeling the semantic relationship between all the evolved web services, into a directed graph. The graph was constructed to find all the shortest paths to optimize the computational efforts related to the web services composition.

Alhadid et al. [25] proposed the Smart Multistage Forward Search (SMFS) as an effective technique to select and construct the service composition. It searched for a web

service with available resources to create a new service composition, even when the web services are integrated within other compositions (to provide more service compositions).

Dongre and Ingle [8] presented their investigation and analysis of the QoS parameters and optimality criteria for services selection and composition. The parameters were: the response time, availability, and reliability. These attributes were the most used ones with minimum/maximum values as optimality criteria.

Researchers proposed several cloud load balancing algorithms and approaches to optimize the VMs' performance and resources utilization. Mishra et al. [38] studied and presented a taxonomy for the load balancing algorithms used in Cloud Computing, which include static and dynamic algorithms. In addition to that, an analysis of the approaches and performance parameters that affect the algorithms was conducted. Chen et al. [39] proposed a dynamic Cloud load balancing (CLB) method to enhance the tasks' scheduling. The proposed model evaluates the VM Load balancing capacity using the computer loading and the server processing power. Lavanya, Vaithiyanathan [40], and Kapur [41] suggested resource scheduling methods to enhance the scalability of the VM. proposed models. They are used to predict the VM workload and to expect the future resource demand to avoid the VM resource overload.

According to the previous discussion, web service selection and construction are still a challenge. Optimization techniques can be adopted to solve the problem of optimizing the web services' selection and service composition. Also, Cloud computing capabilities and the SMO can be utilized to enhance the selection and composition processes and to provide a dynamic solution for resource utilization.

In this research, we will introduce an efficient technique using Cloud computing capabilities and SMO algorithm, to reduce the duration of the web services' selection and composition process and minimize the number of integrated web services. In addition to that, we will also work on enhancing the utilization of the web services' resources and optimizing the resources' reusability effectively and efficiently. The experimental results will be compared with the results of the SMFS technique, to determine the superior algorithm in terms of optimizing service compositions construction processes and utilizing the web service resources.

3. Proposed Model

In this study, we propose a novel service composition model and work on the enhancements implemented on the services' execution engine. The model helps improve the web services' selection and composition processes and optimize the utilization of the web services' available resources. All of that is performed using the SMO algorithm and Cloud computing capabilities. Figure 1 illustrates the process of the proposed model.

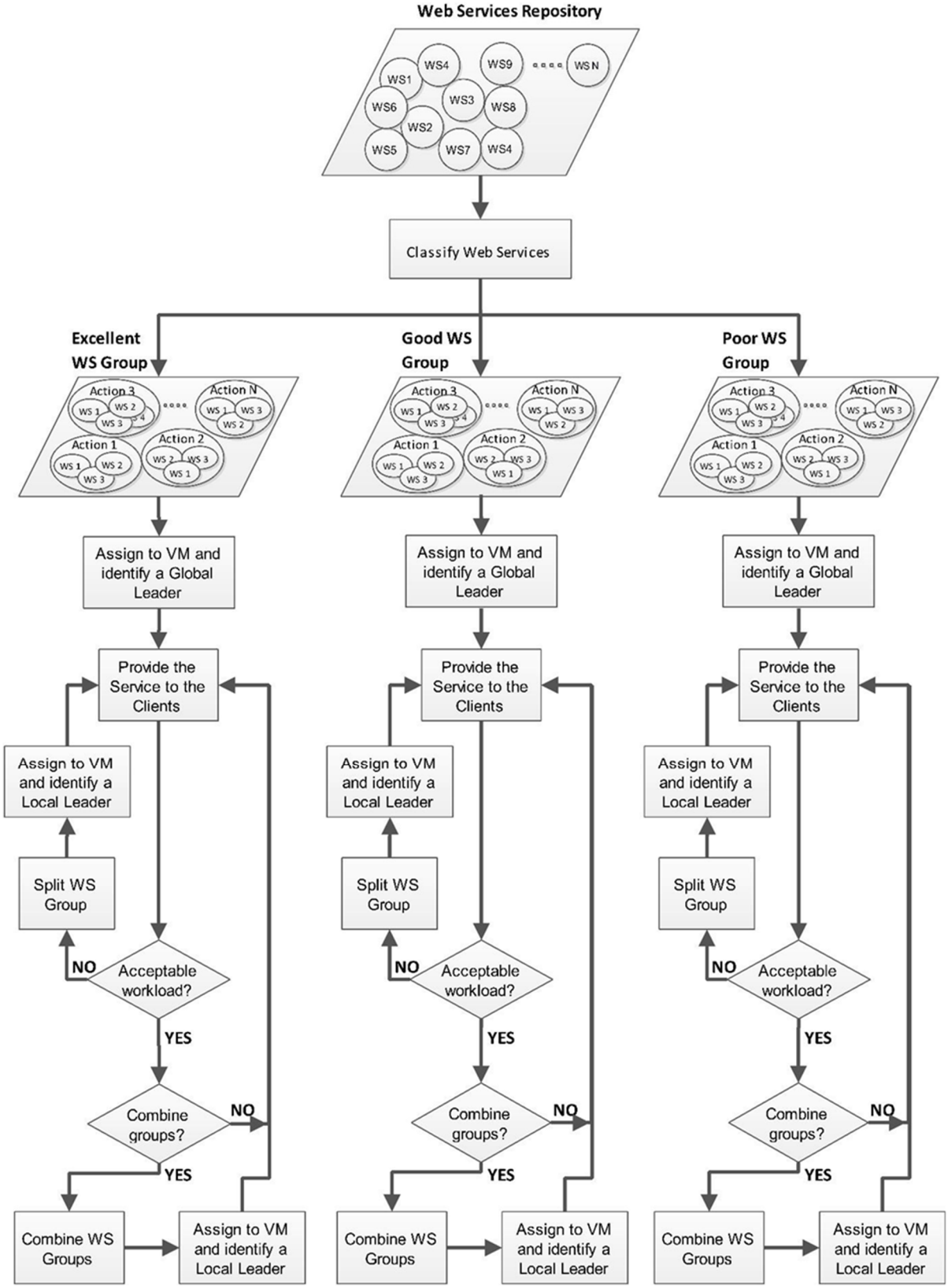

Figure 1. Proposed model processes.

3.1. Step 1. Initialize

During this step, the following must be identified: the number of the Web service repository, number of clients, clients' SLA requirements, clients' classes, virtual machines,

and mapping between each VM and clients' classes. In the initial state, the set of Virtual Machines (VMs) includes only one VM, which will be used to evaluate the web services. It will be split into more than one VMs according to the number of web service classes. The set of VM for each web service class created (related to the number of VMs) will be empty until the related VMs are created.

Set of Web Services: WS = {WS1, WS2, . . . ,WSn}
Set of clients: CL = {CL1, CL2, . . . , CLm}
Set of Clients' SLA Requirements: CLR = {CLR1, CLR2, . . . , CLRL}
Set of Clients' Classes: CLC = {CLC1, CLC2, . . . , CLCx}
Set of Virtual Machines VMs = {VM1}
Set of VM for each CLC: CLC_VM = { }

3.2. Step 2. Evaluate the Population

The web services, located in the repository, are classified on two levels. First, they are classified according to the web services' QoS attributes which are: the response time, the Throughput, and the availability. The QoS attributes are used to find the Reward Function of the web service and the service compositions. Next, the web services are grouped into different classes according to their actions, which represent the functionality of each web service.

The total time needed for a web service to successfully respond to a request is referred to as "web service response time". Table 1 shows the response time related to the web services classifications [12,33,42].

Table 1. Classifications of the web services response time adapted from [12,33,42].

Class	Web Services
Excellent	≥0.1–0.5 s
Good	>0.5–1.0 s
Poor	>1.0–3 s

Web Service response time can be calculated using the following equation, proposed by Emeakaroha et al. [43]:

$$RespTime = RespTime\ (in) + RespTime\ (out) \quad (1)$$

where

$$RespTime\ (in) = \frac{packetsize}{available\ bandwidth - in\ bytes} \quad (2)$$

$$RespTime\ (out) = \frac{packetsize}{available\ bandwidth\ in - out\ bytes} \quad (3)$$

Another QoS attribute related to web services is throughput. Throughput is the number of requests that the web service can process successfully within a defined period. Table 2 shows the web services' throughput classifications [44].

Table 2. Web services' throughput classifications adapted from [44].

Class	Value
Excellent	1800 req./s
Good	1600 req./s
Poor	1000 req./s

Web services throughput can be calculated using the following equation [39]:

$$Thr = \frac{Cs}{Time} \quad (4)$$

where:

- *Cs* is the number of web service requests served successfully.
- *Time* is the period of the web service's operation being investigated.

The web service's availability is considered as one of the most important QoS attributes. It measures the probability of the web service's accessibility and readiness for immediate use, to deliver the service when demanded [43]. Table 3 represents the web services' availability classifications.

Table 3. SAL, web services' availability classifications adapted from [12,42].

Class	Value
Excellent	99–100%
Good	97–98%
Poor	94–96%

Web services' availability can be calculated using the following equation [43]:

$$\text{Ava} = \frac{up\ time}{Total\ Time} * 100\% \tag{5}$$

where:

- Up *time* is the web service's operational *time* between the last web service failure to the next failure, which is also known as the mean *time* between failure (MTBF).
- Total *time* is the overall period of the web service's operation being investigated.

Reward Functions ($R1$ and $R2$) are used to find the web service's QoS reward, the values of $R1$ and $R2$ are gained using the web service's QoS attributes. The goal of calculating the $R1$ and $R2$ is to provide the web service with the maximum availability and throughput, while keeping the minimum response time considering the SLA constraints and requirements. In addition, $R1$ and $R2$ values provide which attribute affects the web service QoS. $R1$ and $R2$ are employed to confirm that the web service achieves an acceptable range for the values of availability, response time, and throughput, and as a result, $R1$ and $R2$ values are used to enhance the process of web service classification.

The value of $R1$ is calculated by the following equation:

$$R1(WSi) = \alpha - \beta + \gamma \tag{6}$$

While the value of $R2$ is calculated by the following equation:

$$R2(WSi) = \gamma - \beta \tag{7}$$

where:

- $R(WSi)$: is the reward value gained by the web service (i).
- α: is the throughput reward value related to the web service (i).
- β: is the response time reward value related to the web service (i).
- γ: is the availability reward value related to the web service (i).

The value of the web service's throughput reward (α) can by calculated using:

$$\alpha = \frac{Thr^s - Thr^{min}}{Thr^{max} - Thr^{min}} \tag{8}$$

where:

- Thr^S: The throughput value of the web service with state (i) calculated using Equation (4).
- Thr^{max}: The maximum value of the web service's throughput in the web services' class.
- Thr^{min}: The minimum value of the web service's throughput in the web services' class.
- While the value of the web service's response time reward (β) can be calculated using:

$$\beta = \frac{RespTime^s - RespTime^{min}}{RespTime^{max} - RespTime^{min}} \tag{9}$$

where:

- $RespTime^S$: The response time value of the web service with state (i) calculated using Equation (1).

- $RespTime^{max}$: The maximum value of the web service's Response Time in the Web Services' class.
- $RespTime^{min}$: The minimum value of the web service's Response Time in the Web Services' class.
- Finally, the value of the web service's availability reward (γ) can be calculated using:

$$\gamma = \frac{Ava^{s} - Ava^{min}}{Ava^{max} - Ava^{min}} \tag{10}$$

where:

- Ava^{S}: The availability value of the web service with state (i) calculated using Equation (5).
- Ava^{min}: The minimum value of the web service's availability in the web services' class.
- Ava^{max}: The maximum value of the web service's availability in the web services' class.

After completing the required calculations, we will have three of the web services classes; Excellent, Good, and Poor. Each class includes all the web services within the same QoS class that can perform all tasks to carry out the required business process, which represents the service composition.

The values of $R1$ and $R2$ are used to show the effect of the QoS attributes, if there is a remarkable difference between the $R1$ and $R2$ values then the clients' SLA requirements might not be satisfied. The value of the reward function ($R2$) is related to the web service's response time and availability QoS attributes while the value of the reward function $R1$ is influenced by the $R2$ QoS attributes in addition to the throughput of the web service. If the value of the $R1$ is greater than the value of $R2$ and the difference is remarkable, this means that web services throughput is high, and we must study the other QoS attributes to make sure that it will be affect the SLA requirements and we must check the values of the QoS attributes before the classification process. Otherwise, if $R1$ and $R2$ have the same values, this shows that the values of the QoS attributes within the same range and the web service can be classified according to the value of the $R1$. For example, if the value of $R1$ = 0.99 while the value of $R2$ = 0.83, this means that the web service's throughput has a significant effect on the values of the reward functions. Table 4 shows the QoS attributes' values of the mentioned example.

Table 4. QoS attributes' values.

Response Time	Availability	Throughput	γ	β	α	$R1$	$R2$	$R1$–$R2$
302.75	89	7.1	0.88	0.054	0.16	0.99	0.83	0.16

The values in Table 4 confirm that the web services' throughput might influence the reward function $R1$ value and give a misleading value related to the web service QoS, and as a result, the SLA may not be achieved.

According to the web service action, we build the service composition. If the web service's action value equals 1, then it represents the first task performed in the service composition, while the web service with the action value equals 2, which represents the second task. This continues until the last action, which is the last task in the service composition. The number of actions is decided by the service composition that shows the business process and the web services found in the dataset. To perform this at a quicker pace, we group the web services according to their action after completing the first level of classification (where the reward function is calculated). More information on the usage of service actions will be explained below.

Service composition reward value can be calculated using the Equation (11).

$$\mathrm{SCr} = \sum_{i=1}^{n} R_x(WSi) \tag{11}$$

where:

- $R_x(WSi)$: Where Rx is the value of the reward function $R1$ if there is no significant difference between the values of $R1$ and the value of $R2$. If there is a significant and remarkable difference between the values of $R1$ and $R2$ then $R2$ will be considered as the value of Rx in Equation (11). Where $R1$ is gained using the Equation (6), while $R2$ is calculated using Equation (7).

3.3. Step 3. Identify Local Leaders for Each Web Service's Class

For each group, assign a Leader by creating a VM and service Load Balancer Object from the Service Load Balancer Class. The Leader handles all the requests dispatched from the service execution engine. So, a set of virtual machines will be changed based on the number of classes. The set of VM for each web service class will be updated according to the number of VMs created.

- Set of Virtual Machines VMs = {VM1}
- Set of VM for each CLC: CLC_VM = {{ CLC1_VM1 }, { CLC2_VM1}, {CLC3_VM1}}

3.4. Step 4. Position Update by Local Leader Phase and Construct Service Compositions Groups

Three service composition groups are constructed using the MFS algorithm. Web services related to each class of clients will be assigned to a global leader, to decrease the time complexity from $O(N_2)$ to $O(N_1)2+ O(N_2)^2 + O(N_3)^2$ using the virtualization technology. N_1 is the total number of web services that belong to the excellent class, N_2 is the number of web services with QoS attributes related to the good class, and N_3 is the web services classified with poor QoS specifications. A multistage dependency graph is created for each class group by the VM, where the graph nodes represent the related web services found in the same class. To create the graph, the web services are divided into a set of stages based on their action and output. Meanwhile, the reward function R(WSi) for each node is calculated to find the weight for each node in the graph. Later, the MFS algorithm is implemented to discover the best paths from the source to the target, which signifies the service compositions to be stored on the VM and to be ready for the clients' invocations when the service is requested.

The following equation presents the SMFS' time complexity:

$$T(n) = (N_1)^2 + (N_2)^2 + (N_3)^2 + 3(N)^2 \quad (12)$$

while Equation (12) shows the proposed model's time complexity:

$$T(n) = (N1)^2 + (N2)^2 + (N3)^2 + \left(\frac{N1}{M1}\right)^2 + \left(\frac{N2}{M2}\right)^2 + \left(\frac{N3}{M3}\right)^2 \quad (13)$$

The following Algorithm 1 shows the web service multistage dependency graph construction algorithm.

Algorithm 1. Web Services Dependency Graph's Construction Algorithm.

```
var K = 5/ number of stages in the graph G which represent the business process web services
var i = 1; // stages loop
var T; // number of tuples in graph G
T = 1;
Do until (no more tuple to create)
        Do until (i = k)
                Set all stages values in the tuple to INF; // INF: infinity
          For each WS ∈ WSR // WS is the web service located in the WS repository
                        If WS ∉ G & WSAction = k & WSk ∈ S k-1(L) & WSTempMaxCap > 0
then
                        // Sk ∈ S k-1(L): There are links to the S from the prior S
                        Calculate the R(WS);
                        Assign the R(WS) to the Node;
                        Add WS to graph G;
        Next WS
        K++;
    Loop // Do until (i = k)
                        Loop // next tuple in graph G
```

A service composition is constructed by finding the best path from the source. The path represents the first web service that will be invoked by the service execution engine to the last web service in the business process. The process of construction is carried out by selecting the web services with minimum cost that form the service composition. Algorithm 2 shows the MFS algorithm used to construct the service compositions. Although the service composition with the highest SCr value is the path that provides the best service composition to the client, the workload on the web services and the service compositions must be considered to balance the load on the invoked services during the execution.

Algorithm 2. The proposed MFS Algorithm used to construct the service compositions.

```
var K; // number of stages in 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 Service compositions can be constructed)
    Do until (i = 1)
        Cost(i, j) = Min {C(i, L) + cost (i + 1), L)} where L in Vi + 1 & (j, L) ∈ G.
–i; // prior stage
Loop // stages loop
    Loop // Find next SC
```

Processing capacity PC of *VM(i)* is calculated using the following equation:

$$\mathrm{PC_i} = \mathrm{Pe_i} \times \mathrm{mips_i} \tag{14}$$

where:

- $\mathrm{Pe_i}$: number of processing elements on host.
- $\mathrm{mips_i}$: million instructions per second can be processed by host processors.

On the other hand, the workload can be found using the following equation:

$$\mathrm{Workload}(VMi) = \frac{Tasks\ in\ VM(i) queue \times processing\ time\ for\ each\ task\ in\ the\ queue}{Processing\ Capacity\ (PC) of\ VM(i)}$$

The service composition available capability (SCavaCap) is calculated using the equation:

$$\mathrm{SCavaCap} = \text{service composition max. capability} - \text{current processing requests} \tag{15}$$

After the service compositions construction process, SMO generates uniformly distributed VMs that host the load balancer for each group j where *SM*i represents the ith VM. Each *SM*ij is initialized as follows:

$$SM(\mathrm{i,j}) = SM\mathrm{minj} + \mathrm{U}\ (0,1) + (SM\mathrm{maxj} - SM\mathrm{minj}) \tag{16}$$

where:

- *SM*minj: is the VM with the least acceptable workload and the minimum service composition available capability of the group (j) that meets the minimum SLA expectations. Where *SM*minj equals:

$$SM\mathrm{minj} = \left(\frac{PCi\ [min]}{100}\right) * 50\% + \frac{SCavaCap\ [min]}{100} * 50\% \tag{17}$$

*SM*maxj: is the VM with the most workload and the maximum service composition available capability that can achieve and satisfy the group (j) related to the client's class SLA requirements. Where *SM*maxj equals:

$$SM\mathrm{maxj} = \left(\frac{PCi\ [max]}{100}\right) * 50\% + \frac{SCavaCap\ [max]}{100} * 50\% \tag{18}$$

- U (0,1) is a uniformly distributed random number in the range (0,1).

During the initialization phase, the VMs might have the same processing capacities and service composition available capability. Therefore, the initial VMs' processing capacities and service composition capabilities will be selected to find the values of the SMminj and SMmaxj.

For each calculated *SM*(i,j) value, find the fitness using:

$$\text{Fitness} = \begin{cases} \frac{1}{1+SM\text{ij}} & \text{if } SM\text{ij} \geq 0 \\ 1 + \text{abs}(SM\text{ij}) & \text{if } SM\text{ij} < 0 \end{cases}$$

SM(i,j) with maximum fitness value will be the global leader and the local leader in the initialization phase. Later, the system supplies the services to the clients and gathers information about the VM workload and recourse, in addition to the service compositions and the Web Services QoS attributes' values.

4-The Load Balancer evaluates the workload of the VM (i) by analyzing the hardware systems and measuring the available capabilities and capacity. In addition, the load balancer evaluates the available capability of the service compositions (SCavaCap) and the workload on the integrated web services.

3.5. Step 5: Position Update Process in Local Leader Phase (LLP)

According to the Service Execution Engine evaluation in step (4), the Position update process in the Local Leader Phase (LLP) Algorithm is implemented to update the VM's current state, as shown in Algorithm 3.

Algorithm 3. Position update process in Local Leader Phase (LLP).

```
for each member SMi ∈ kth group do
   for each j ∈ { 1, . . . , D} do
        SM(i,j) = SMminj + U (0,1) + ( SMmaxj − SMminj) . . . . . .
        if SMij ≥ 0
             Fitness = 1/(1+SMij)
        Else
             Fitness = 1 + abs(SMij)
   end for
end for
```

SM(i,j) with the maximum fitness value will be identified as the local leader of the group. This means that the VM with the highest capabilities and available resources is found to be the local leader of the group. The constructed service composition is moved to the new local leader to enhance the provided services.

3.6. Step 6. Learning through Local Leader Learning Phase

According to the QoS attributes' values related to the integrated web services and the service compositions; the system learns about the workload on each class, response time, throughput, and availability, which are used to distribute the load between the different load balancing objects in the same class.

3.7. Step 7: Decide Fission or Fusion

The results of steps (4) and (6) include the updated processing capacity of the VM and the workload of the integrated web services and the service compositions. The results of the load balancer either combine or divide the VM using the virtualization and auto-scaling capabilities or the integrated web services. This aims to improve the services provided and to satisfy the clients' SLA requirements by creating new VMs and/or new service compositions.

Algorithm 4 illustrates the Load Balancer pseudocode and the processes of the group's fission or fusion.

If the VM's workload exceeds 50%, this is considered as a sign that the VMs' processing capacities might soon be exhausted. Accordingly, the load balancer uses the virtualization and auto-scaling capabilities to identify new VMs and divide the web service group. New service compositions are also created to avoid the inability to respond to the clients' requests or reach the maximum capacity of the integrated web services and the service composition. On the other hand, if the VM's workload is equal to or less than 50% while the service composition available capability (SCavaCap) is more than 50% (the same is applicable on

the available VM's resources), the system may not provide the service within the acceptable SLA constraints shortly. In this case, the load balancer uses virtualization and auto-scaling to combine the VMs in the same CLC_VM and merge the web services' groups into one group. Identifying new VM and new load balancer reconstructs new service compositions. Finally, when the VM's workload is less than 50% and the service composition available capability (SCavaCap) is less than 50%, the load balancer keeps the status of the VMs as is, without any upgrades or modifications on the VMs or the service compositions.

The next section shows the experimental results executed using the simulator and real-life data set and the compassion between the results of the proposed model and the result of the SMFS technique, using the service execution engine hosted on three servers.

Algorithm 4. Load Balancer Pseudo Code.

```
For each CLC_VM ∈ CLC_VM set
For each VM ∈ CLC_VM do
if VM_Workload ≤ 50% AND SCavaCap ≤ 50% then
        -  keep the current state of the system.
else if VM_Workload > 50% then // even if the SCavaCap is greater or less than 50%
        -Identify new VM and new Load Balancer; divide the web sevice group and create new
service compositions
        -Create new service compositions using the unused available integrated Web Services'
resources (high max. capacity value)

else if VM_Workload ≤ 50% AND SCavaCap > 50% then
        -Combine VMs in the same CLC_VM and merge the Web Services groups into one group,
identify new VM and new Load Balancer to reconstruct new service compositions.
Loop // Next VM
Loop // Next CLC_VM
```

4. Simulation Results and Evaluation

4.1. Simulator and Dataset

The performance of the proposed method has been evaluated using a simulator programmed using VB.NET. The simulator simulates the WS selection, composition, Spider Monkey optimization behavior, and cloud computing execution. In addition to that, it shows the performance of the web services and the QoS changes and variations. All simulation experiments were performed on an i7-3632QM machine equipped with a 2.20 GHz processor and 8 GB DDR3 RAM. Table 5 shows the initial values of the simulator parameters.

Table 5. Simulator setup parameters.

Parameter	Value
Number of runs	20
Number of clients	1000
Task Length	1–15 sec.
Min. Number of requests/Client	100
Max. Number of requests/Client	5000
Number of VMs	3 (initial)
Number of VMs Processing Elements (PEs)	1
Number of Hosts	1

The simulation dataset has been generated using the real data QWS dataset proposed by Al-Masri and Mahmoud [45]. Generated datasets illustrate different workflow scenarios to evaluate the proposed model's effectiveness and efficiency. The QWS dataset is a labeled dataset describing real-world web services QoS. The dataset includes the web services' response time, throughput, and other QoS results from the results of 2509 web services. For research purposes, the QWS dataset is categorized into three categories: Excellent, Good and Poor web service in all the generated datasets.

4.2. Experimental Results and Discussion

In this section, we compare the simulation runs' results of the proposed model and the results of the SMFS technique [25]. They are hosted on three VMs without virtualization and auto-scaling capabilities, to decide the superior algorithm in terms of optimizing service compositions construction processes, in addition to the use of web service resources. The comparison between the results is based on the SLA clients' classes, which are excellent, good, and poor classes.

Figure 2 shows the number of integrated web services in the service composition, the figure shows that the resources' use of the web services using the proposed model is better than the other models. Where all the web services joined the SMO groups and integrated into service compositions in all web services classes. The results show that using SMO enhances the average number of the integrated web services in the service composition by 11.7%, in comparison with the results of the Simulated Annealing SMFS technique. The enhancement is a result of sharing resources by web services in the same group, where the web service might be integrated into more than one service composition at the same time according to the maximum capacity of the web service.

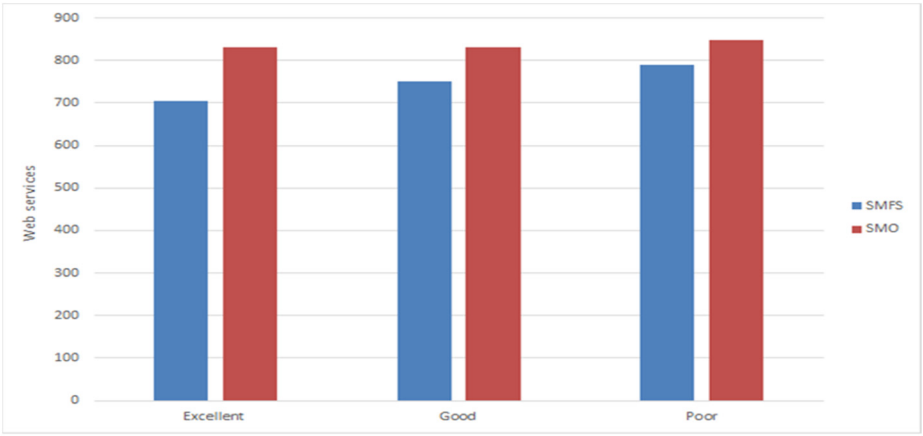

Figure 2. Number of the integrated web services in the service composition.

Figure 3 illustrates the comparison between the number of constructed service compositions using the SMO and SMFS. The results show that the number of constructed service compositions by the SMFS is greater than that of the proposed SMO model by 11.8%. This SMFS technique searches the available web services and integrated web services for available resources, to construct new service compositions. As a result of the engagement of the web services with groups in the proposed SMO model, two or more web services with the same functionality might integrate within the same service composition to supply the service—this is related to the maximum capacity of the web services and the number of the client's requests that the web service can handle per second, where each service composition consists of at least five web services to provide an end-to-end full service.

Figure 4 shows that the proposed model improves the composition time. The results presented in Figure 4 confirm that the time needed by the SMFS technique during the construction of the composition services is longer than the time needed by the proposed model using SMO. The composition time required by the SMFS is 1645.71 s, while the time necessary to complete the composition process by the SMO proposed model is 1172.03 s.

Moreover, the comparison of the proposed model and the SMFS technique's throughput is shown in Figure 5, the dynamic behavior of the proposed model's SMO enhances the system throughput. The results show that the average number of requests successfully processed by the service compositions within the acceptable SLA response time increased by 24.8%, compared to the throughput of the SMFS technique within the acceptable SLA response time for all client classes.

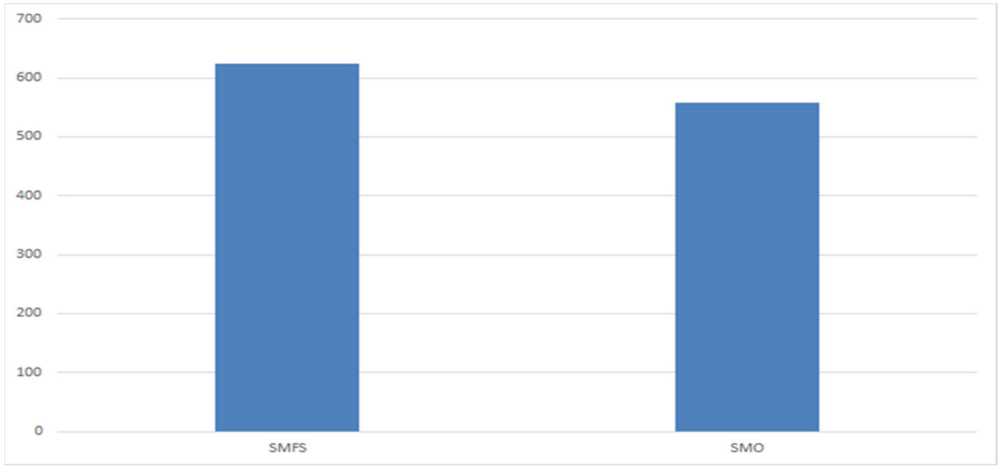

Figure 3. Number of the Constructed Service Compositions.

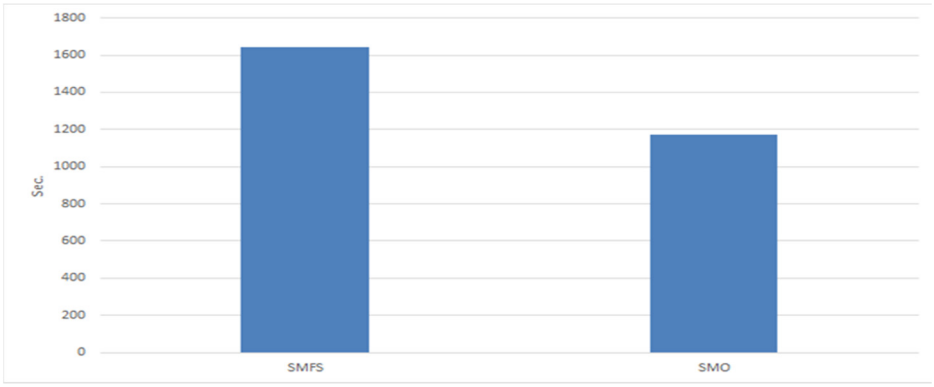

Figure 4. Service composition time.

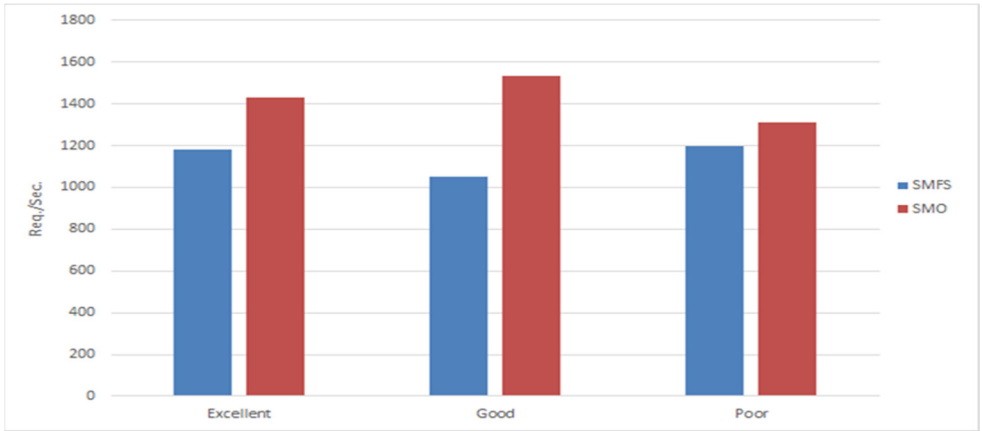

Figure 5. Throughput.

Finally, Figure 6 shows the results of the service composition response time related to the proposed model and the SMFS technique. The R\response time results for both models are within the acceptable range for all classes.

Figure 6 confirms that the proposed model's response time is better than that of the SMFS technique (for all classes); where the response time of the proposed model is less than the SMFS' by 0.25, 0.69, and 5.35 s for the Excellent, Good and Poor classes, respectively.

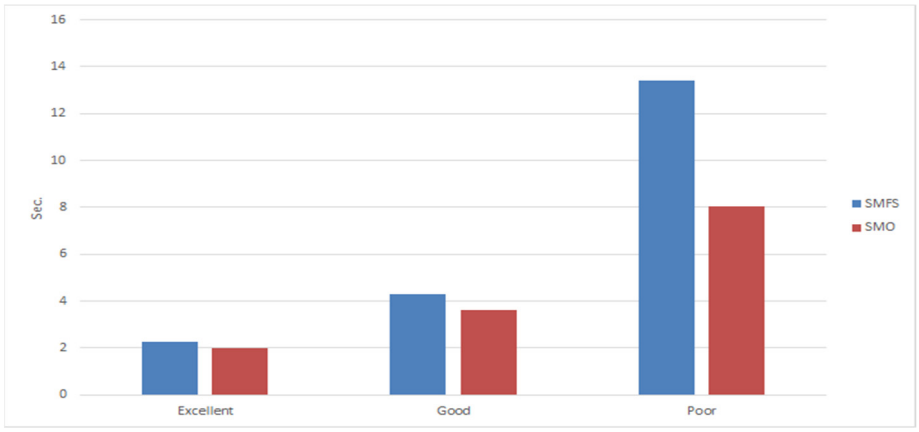

Figure 6. Service compositions' Response time.

5. Conclusions

The experimental results show that using the proposed SMO model decreases the average composition time by 40.4% compared to the composition time required by the SMFS technique, where the Equation (12) presents the SMFS' time complexity, while Equation (13) shows the proposed model's time complexity

In both Equations (11) and (12), the variable N is the number of all web services found in the repository. N1 is the total number of web services that belong to the excellent class, N2 is the number of web services with QoS attributes related to the good class, and N3 is classified as web services with poor QoS specifications. While the variables M1, M2, and M1 in Equation (12) are the number of web service groups of the 1st, 2nd, and 3rd VMs, respectively.

According to Equations (11) and (12), the initial time complexity for both models equal $(N1)^2$, $(N2)^2$, $(N3)^2$ defined and described in the previous paragraph. During the execution of the proposed model, the time complexity changed to $(\frac{N1}{M1})^2$, $(\frac{N2}{M2})^2$ and $(\frac{N3}{M3})^2$, because of dividing the web services' groups N1, N2, and N3 into the number of groups M1, M2, and M3 respectively.

On the other hand, $3(N)^2$ is the extra time needed to search and construct new service compositions in the multi-classes pool by the SMFS technique during the execution, which confirms the results of the simulation mentioned in Section 4.2. The experimental results also show that the web services joined the SMO groups and integrated into service compositions, SMO increased the average number of integrated web services in the service composition by 11.7%, in comparison with the results of the SMFS technique.

The results also show that the dynamic behavior of the SMO improves the proposed model throughput. The average number of requests that the service compositions processed successfully increased by 1.25% compared to the throughput of the SMFS technique. In addition to that, the proposed model decreases the Service compositions' response time by 0.25 s, 0.69 s, and 5.35 s for the Excellent, Good and Poor classes respectively, compared to the results of the SMFS Service compositions' Response time related to the same classes. On the other hand, the number of constructed service compositions by the SMFS is greater than the one achieved using the proposed SMO model by 11.8%. It is a result of integrating the web services into more than one service composition and using the available resources to construct new service compositions.

One of the drawbacks of the proposed model is the machines' hardware specifications, which may directly affect the efficiency of the cloud system and the services provided if the machines' hardware is not appropriate for the required workload. Furthermore, the availability of the web service that provides different functionalities and belongs to different classes in the repository is one of the major challenges to construct the service compositions. It is considered a drawback because when the web service with the required functionality is not available, it will be difficult to construct the service composition. The

service load balancer will not be able to achieve the required SLA when the workload is high, or in case there is no substitutional web service when the web service is down or cannot offer the service with the minimum SLA requirements.

In the future work, we suggest adopting cloud services to enhance the model by using the virtualization and auto-scaling capabilities, which will increase the number of web service threads with the same function and will provide the service to a substantial number of clients with the same SLA constraints and requirements.

Author Contributions: Conceptualization, I.A. and H.T.; methodology, I.A.; soft-ware, I.A.; validation, H.T., I.A. and S.A.; investigation, S.A.; resources, I.A.; data curation H.T.; writing—original draft preparation, I.A.; writing—review and editing, S.K.; visualization, H.T. and I.A.; supervision, S.K.; project administration, S.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zeng, L.; Benatallah, B.; Dumas, M.; Kalagnanam, J.; Sheng, Q.Z. Quality driven web services composition. In Proceedings of the 12th International Conference on World Wide Web, New York, NY, USA, 20 May 2003; pp. 411–421.
2. Dongre, Y.V.; Ingle, R.B. QoS Based Optimal Resource Allocation in Service Composition for Heterogeneous Devices. In Proceedings of the 2019 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 17–19 July 2019; pp. 763–767. [CrossRef]
3. Mathew, G.E.; Shields, J.; Verma, V. QoS based pricing for web services. In *International Conference on Web Information Systems Engineering*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 264–275.
4. Zhao, X.; Li, R.; Zuo, X. Advances on QoS-aware web service selection and composition with nature-inspired computing. *CAAI Trans. Intell. Technol.* **2019**, *4*, 159–174. [CrossRef]
5. Dongre, Y.; Ingle, R. An Investigation of QoS Criteria for Optimal Services Selection in Composition. In Proceedings of the 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, India, 5–7 March 2020; pp. 705–710.
6. Uc-Cetina, V.; Moo-Mena, F.; Hernandez-Ucan, R. Composition of web services using Markov decision processes and dynamic programming. *Sci. World J.* **2015**, *2015*, 545308. [CrossRef] [PubMed]
7. Afaneh, S.; Alhadid, I. Airport enterprise service bus with three levels self-healing architecture (AESB-3LSH). *Int. J. Space Technol. Manag. Innov.* **2013**, *3*, 1–23. [CrossRef]
8. AlHadid, I.; Abu-Taieh, E. Web Services Composition Using Dynamic Classification and Simulated Annealing. *Mod. Appl. Sci.* **2018**, *12*, 376–386. [CrossRef]
9. Rai, G.N.; Gangadharan, G.R.; Padmanabhan, V.; Buyya, R. Web service interaction modeling and verification using recursive composition algebra. *IEEE Trans. Serv. Comput.* **2018**, *14*, 300–314. [CrossRef]
10. Jatoth, C.; Gangadharan, G.; Fiore, U.; Buyya, R. QoS-aware big service composition using MapReduce based evolutionary algorithm with guided mutation. *Future Gener. Comput. Syst.* **2018**, *86*, 1008–1018. [CrossRef]
11. Zheng, Z.; Zhang, Y.; Lyu, M.R. Investigating QoS of Real-World Web Services. *IEEE Trans. Serv. Comput.* **2014**, *7*, 32–39. [CrossRef]
12. Wang, C.; Ma, H.; Chen, G.; Hartmann, S. A memetic NSGA-II with EDA-based local search for fully automated multi objective web service composition. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, 13–17 July 2019; pp. 421–422.
13. Juric, M.B.; Mathew, B.; Sarang, P.G. *Business Process Execution Language for Web Services: An Architect and Developer's Guide to Orchestrating Web Services Using BPEL4WS*; Packt Publishing Ltd.: Birmingham, UK, 2006.
14. Muthusamy, V.; Jacobsen, H.-A.; Chau, T.; Chan, A.; Coulthard, P. SLA-driven business process management in SOA. In Proceedings of the 2009 Conference of the Center for Advanced Studies, Toronto, ON, Canada, 2–5 November 2009; pp. 86–100.
15. Fan, S.-L.; Yang, Y.-B.; Wang, X.-X. Efficient web service composition via knapsack-variant algorithm. In *International Conference on Services Computing*; Springer: Cham, Switzerland, 2018; pp. 51–66.
16. Mirzayi, S.; Rafe, V. A hybrid heuristic workflow scheduling algorithm for cloud computing environments. *J. Exp. Theor. Artif. Intell.* **2015**, *27*, 721–735. [CrossRef]

17. Dubey, A.; Pal, S. *Dynamic Service Composition towards Database Virtualization for Efficient Data Management*; IEEE: New York, NY, USA, 2017; ISBN 978-1-5090-3519-9.
18. Chen, L.; Xian, M.; Liu, J.; Wang, H. Research on Virtualization Security in Cloud Computing. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *806*, 012027. [CrossRef]
19. Singh, P.; Kaur, A.; Gupta, P.; Gill, S.S.; Jyoti, K. RHAS: Robust hybrid auto-scaling for web applications in cloud computing. *Clust. Comput.* **2021**, *24*, 717–737. [CrossRef]
20. Al Rawajbeh, M. Performance evaluation of a computer network in a cloud computing environment. *ICIC Express Lett.* **2019**, *13*, 719–727.
21. Sharma, H.; Hazrati, G.; Bansal, J.C. Spider monkey optimization algorithm. In *Evolutionary and Swarm Intelligence Algorithms*; Springer: Cham, Switzerland, 2019; pp. 43–59.
22. Mousa, A.; Bentahar, J. An efficient QoS-aware web services selection using social spider algorithm. *Procedia Comput. Sci.* **2016**, *94*, 176–182. [CrossRef]
23. Sangaiah, A.K.; Bian, G.-B.; Bozorgi, S.M.; Suraki, M.Y.; Hosseinabadi, A.A.R.; Shareh, M.B. A novel quality-of-service-aware web services composition using biogeography-based optimization algorithm. *Soft Comput.* **2020**, *24*, 8125–8137. [CrossRef]
24. Emeakaroha, V.C.; Brandic, I.; Maurer, M.; Dustdar, S. Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS), Caen, France, 28 June–2 July 2010; pp. 48–54.
25. Karimi, M.; Esfahani, F.S.; Noorafza, N. Improving response time of web service composition based on QoS properties. *Indian J. Sci. Technol.* **2015**, *8*, 1–8. [CrossRef]
26. Jung, J.; Krishnamurthy, B.; Rabinovich, M. 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, Honolulu, HI, USA, 7–11 May 2002; pp. 293–304.
27. Gao, Y.; Na, J.; Zhang, B.; Yang, L.; Gong, Q. Optimal web services selection using dynamic programming. In Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06), Cagliari, Italy, 26–29 June 2006; pp. 365–370.
28. Yau, D.K.Y.; Lui, J.C.S.; Liang, F. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. *IEEE/ACM Trans. Netw.* **2005**, *13*, 29–42. [CrossRef]
29. Tilahun, S.L.; Ngnotchouye, J.M.T.; Hamadneh, N. Continuous versions of firefly algorithm: A review. *Artif. Intell. Rev.* **2017**, *51*, 445–492. [CrossRef]
30. Shree, S.U.; Amuthan, A.; Joseph, K.S. Integrated Ant Colony and Artificial Bee Colony Optimization Meta Heuristic Mechanism for Quality of Service Based Web Service Composition. *J. Comput. Theor. Nanosci.* **2019**, *16*, 1444–1453. [CrossRef]
31. Jung, J.-Y.; Bae, J.; Liu, L. Hierarchical clustering of business process models. *Int. J. Innov. Comput. Inf. Control* **2009**, *5*, 1349–4198.
32. Gao, Z.-P.; Chen, J.; Qiu, X.-S.; Meng, L.-M. QoE/QoS driven simulated annealing-based genetic algorithm for Web services selection. *J. China Univ. Posts Telecommun.* **2009**, *16*, 102–107. [CrossRef]
33. Elmaghraoui, H.; Zaoui, I.; Chiadmi, D.; Benhlima, L. Graph based E-Government web service composition. *arXiv* **2011**, arXiv:1111.6401.
34. Mishra, S.K.; Sahoo, B.; Parida, P.P. Load balancing in cloud computing: A big picture. *J. King Saud Univ.-Comput. Inf. Sci.* **2020**, *32*, 149–158. [CrossRef]
35. Androcec, D.; Vrček, N.; Küngas, P. Service-Level Interoperability Issues of Platform as a Service. In Proceedings of the 2015 IEEE World Congress on Services, New York, NY, USA, 27 June–2 July 2015; pp. 349–356.
36. Chen, S.L.; Chen, Y.Y.; Kuo, S.H. CLB: A novel load balancing architecture and algorithm for cloud services. *Comput. Electr. Eng.* **2017**, *58*, 154–160. [CrossRef]
37. Lin, Z.; Zhao, H.; Ramanathan, S. Pricing Web Services for Optimizing Resource Allocation—An Implementation Scheme of the 2nd Workshop on e-Business, Seattle, WA, USA, 13–14 December 2003. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.7799&rep=rep1&type=pdf (accessed on 6 December 2021).
38. Al Rawajbeh, M.; Al Hadid, I.; Aqaba, J.; Al-Zoubi, H. Adoption of cloud computing in higher education sector: An overview. *Indian J. Sci. Technol.* **2019**, *5*, 23–29.
39. Al-Masri, E.; Mahmoud, Q.H. Toward quality-driven web service discovery. *IT Prof.* **2008**, *10*, 24–28. [CrossRef]
40. Karunamurthy, R.; Khendek, F.; Glitho, R.H. A novel architecture for Web service composition. *J. Netw. Comput. Appl.* **2012**, *35*, 787–802. [CrossRef]
41. Alhadid, I.; Tarawneh, H.; Kaabneh, K.; Masa'Deh, R.; Hamadneh, N.N.; Tahir, M.; Khwaldeh, S. Optimizing Service Composition (SC) Using Smart Multistage Forward Search (SMFS). *Intell. Autom. Soft Comput.* **2021**, *28*, 321–336. [CrossRef]
42. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; et al. A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58. [CrossRef]
43. Lavanya, M.; Vaithiyanathan, V. Load prediction algorithm for dynamic resource allocation. *Indian J. Sci. Technol.* **2015**, *8*, 1–4. [CrossRef]
44. Ludwig, H.; Keller, A.; Dan, A.; King, R.P.; Franck, R. *Web Service Level Agreement (WSLA) Language Specification*; IBM Corporation: Armonk, NY, USA, 2002.
45. AlHadid, I.; Kabbaneh, K.; Tarawneh, H.; Alhroob, A.; Abu-Taieh, E.; Khwaldeh, S.; Alrwashdeh, D.; Alkhawaldeh, R.S. Adaptive Methods to Optimize Web Services Selection and Service Compositions Construction. *New Ideas Concern. Sci. Technol.* **2021**, *8*, 74–86.