EFFECTIVE SERVICE COMPOSITION APPROACH BASED ON PRUNING PERFORMANCE BOTTLENECKS

Navinderjit Kaur Kahlon and Kuljit Kaur Chahal

Department of Computer Science
Guru Nanak Dev University, Amritsar India

ABSTRACT

In recent years, several online services have proliferated to provide similar services with same functionality by different service providers with varying Quality of Service (QoS) properties. So, service composition should provide effective adaptation especially in a dynamically changing composition environment. Meanwhile, a large number of component services pose scalability issues. As a result, monitoring and resolving performance problems in web services based systems is challenging task as these systems depend on component web services that are distributed in nature. In this paper, a distributed approach is used to identify performance related problems in component web services. The service composition adaptation provides timely replacement of the performance bottleneck source that can prohibit performance degradation for the forthcoming requests. Experimentation results demonstrate the efficiency of the proposed approach, and also the quality of solution of a service composition is maintained.

KEYWORDS

Web service composition; Quality of Service; reconfiguration; self-adaptive; optimal.

1. INTRODUCTION

A web services based software system also referred as a Composite Web Service (CWS) makes use of third party web services which run on-the-fly. Adaptability is one of the key requirements of such systems keeping in view the dynamic environment in which they execute [1]. Despite having to be continuously available, they also need to strive to remain optimal in changing conditions. Thus, reconfiguration of a CWS is required to adjust or adapt as per the changed scenario [2].So an important infrastructure level concern is to add self-adaptation ability so that a web services based solution can adapt seamlessly as the execution environment changes.

A self-adaptive (self-healing or autonomic) system possesses capabilities to reconfigure in response to changing environment conditions [3]. In a self-adaptive system, monitoring occurs alongside its execution. The system may need to adapt depending upon the information extracted during monitoring. However, monitoring and adaptation [4] is not trivial in a web services based system which is distributed at different physical locations. The distributed ownership of the component web services add to the complexity of such systems.

Most of the existing approaches that focus on the challenges of building complex web services based systems; treat critical situations arising in a dynamic execution environment as exceptions [5] or failures that require repair [6]. However a web service based system executing in a static environment cannot be a real world application. In a real world web services based system;

change is not an exception but a rule. Changes are a natural phenomenon for such systems as they operate in a dynamic execution environment. Moreover, a web services based system has not only to be correct and reliable, but it should also ensure that it remains optimal as the component web services evolve independently. The need to reconfigure a web service composition in a timely manner when change events happen during its execution is an important issue and still an open question [7, 8].

In this paper, a distributed (a hybrid of client and server) monitoring approach that keeps track of QoS values of different component web services of a web services based system is presented. Whenever QoS of a component web service degrades, the composite web service is notified which then replaces the component web service with an alternative. We extend this framework further to identify component web services which make the composite web service sub-optimal. Such web services are pruned and replaced with better alternatives. Better alternatives may be available in distant/premium service repositories. It is assumed that all web services are substitutable and it is feasible to find direct substitutes either in local or in distant/premium repositories.

The main contributions of this paper are:

- The service clients are notified just-in-time using publish-subscribe mechanism when QoS of a component web service degrades. The service client then automatically (without any human intervention) shifts to a better alternative.
- To localize the source of performance bottlenecks by monitoring the present workflow and pruning such source by replacing it with a better alternative in the proposed framework.

This paper contains five sections. Section 2 presents related work about web service composition and self-adaptive web service composition at runtime. Section 3 gives an overview of the system design, and different modules to implement the proposed approach. Section 4 presents the results of the experiments. Finally, section 5 concludes the paper.

2. Related Work

Existing web service composition approaches [9,10,11] strive to find an optimal solution at design phase, which is not efficient (as it is a NP-hard problem) and does not scale up for a large data set size. It also does not reflect a real world situation where a web service composition has to remain optimal in a dynamic execution environment. As consumers find it difficult to select a service composition that is near optimal solution satisfying functional and non-functional requirements, eagle strategy can be used [27].

Liu et al. [12] use prediction based on case based reasoning to solve web service composition problem efficiently and effectively. Moustafa and Zang [13] predict potential degradation scenarios, and apply a proactive approach whenever the system deviates from expected QoS. Such solutions are appropriate for a dynamic execution environment. But their major drawback is that many a time predictions fail, and the overhead incurred to handle failed prediction may be high. In case of the Internet based applications, QoS degradation of component web services may be transient when the system load is high at one point of time. The dynamic optimization of a service composition can be done by using multi-agent reinforcement learning [26]. A distributed Q-learning algorithm is used to accelerate the convergence rate.

Angarita et al. [14] propose to build fault tolerant CWS to ensure that either such a CWS completes successfully or leaves the execution in a safe state when component web services fail to perform as per expectations. Campos et al. [8] present an approach for building adaptive service compositions by detecting undesirable behaviors in their execution traces. They propose to use formal methods to verify web service adaptation at execution time in a dynamic environment.

After detecting violations in QoS values, the adaptation mechanism gets triggered. Adaptation involves reconfiguring the execution plan without stopping the composite service execution. Adaptation may be implemented by following a reactive [8], a proactive approach [15], or a post-mortem of the previous execution traces to improve the system for future [16].

Zhu et al. [17] argue that effective runtime adaptation of service needs real changes in QoS of web services for timely and accurate decisions about -When to trigger adaptation action?, Which web service to replace in execution?, and Which candidate web service to select? Adaptable Web Services Framework (AWSF) [18] and Self ADaptIve for web service Compositon (SADICO)[19] are the two frameworks which take into consideration the service user's context (e.g. device features such as screen size, bandwidth, or user location) and adapt the web service behavior so that web service becomes more relevant and useful.

Although, many solutions have been proposed to adapt a web services based software solution to QoS changes of its component web services [20,21], but runtime monitoring of component web services, and then communication of the data, collected during monitoring, to the clients still needs to improve.

3. System Design And Implementation

3.1 Assumptions

This work relies on a few key assumptions. First, it assumes that all component web services are substitutable and it is feasible to find direct substitutes either in local or in distant/premium repositories. Second, there exist two distinct periods of performance i.e. execution traces with distinct levels of performance. Third, the workload (i.e. request arrival rate) is uniform across different periods of analysis. Performance comparison of different execution traces under different workloads is not justified. Fourth, all the component web service are supposed to have same levels of performance. Though it seems unrealistic, but this assumption is motivated by the quality characteristics of web services. Fifth, the network connection between web services is error free, even though individual atomic services may be problematic. Lastly, service clients and providers are trustworthy entities, and there are no security risks when mobile agents execute on the provider side.

3.2 System Overview

This section presents the basic components of our proposed system by describing their own functions along with their interaction with other components in the system.
The proposed system is composed of five components as:

A. The Basic Process

A workflow manager receives a client request; gets the corresponding abstract composition; selects corresponding concrete web services; dispatches mobile agents to service providers to monitor the QoS behavior of the chosen web services for execution in composition; invokes the partner web services for preparing the results and responds back to the client.

B. Monitoring the logs for QoS degradation

Once the deployed web services are invoked upon client request in the composition workflow, continuous monitoring and analysis of component web services is done by examining their execution logs on the provider side. The execution logs consisting of QoS values of web services are maintained at the service provider side when each web service gets executed in the execution workflow. Each web service execution log is maintained for future prediction based analysis. Monitoring and adaptation of the component web services is based on four QoS parameters as execution time, throughput, reliability and availability. The values for QoS parameters are calculated using the formulae in Table 1.

Table1: QoS Parameters formulae

Execution Time	Response Time – Request Time
Throughput	$\frac{\text{No of completed requests}}{\text{unit time}}$
Reliability	$\frac{\text{Number of failures}}{\text{total requests per unit time}}$
Availability	$\frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$

The throughput, reliability, and availability are the attributes with positive dimension i.e. higher the value, better it is, whereas, execution time is a negative dimension.

C. Log Monitoring to identify the source of performance bottleneck

This section presents an approach to improve QoS of a composite web service when some of its component web services are acting as a performance bottleneck in the service composition. Inter Quartile Range (IQR) is computed to measure variability in the data set by analyzing the previous execution logs. IQR is the difference between first quartile (Q1) and third quartile (Q3). The upper and lower threshold values in the dataset corresponding to every QoS attributes are calculated by using Tukey Fences [22] method which is a popular method of identifying extreme data values in a data set.

$$\text{Lower threshold} = Q1-1.5(IQR) \qquad \text{----}(1)$$
$$\text{Upper threshold} = Q3+1.5(IQR) \qquad \text{----}(2)$$

A component web service executing in the workflow that can act as performance bottleneck can be detected based on its QoS values. The comparison against the threshold values depend upon the type of dimension of the QoS attributes. A component web service with QoS attribute as a negative (positive) dimension will be compared with the upper (lower) threshold value. In case there are multiple component web services adding to the performance degradation of the workflow, then pruning of a web service is decided on the basis of the quantum of influence that a web service has on the workflow. A web service with maximum influence is pruned first and then the others in that order.

A workflow in a service composition may follow serial, cyclic, parallel, or a combination of the three execution patterns of partner web services [23]. Aggregate value of a QoS attribute for a CWS is calculated using different formulae for the different workflow patterns. Table 2 gives the formulae for a sequential workflow used in the service composition. In a sequential pattern, the component web services execute in a serial order. The proposed service composition is a simple sequence of service executions.

Table 2: QoS aggregate formulae for a sequential workflow

QoS Constraint	Sequence
Execution Time	$\sum_{i=1}^{n}\sum_{j=1}^{l_i} q_{et}(c_{ij}).$
Throughput	$Min \sum_{i=1}^{n}\sum_{j=1}^{l_i} q_{tp}$
Reliability	$\prod_{i}^{n} q_{rl}(ci)$
Availability	$\prod_{i}^{n} q_{al}(ci)$

D. Just-in-time Notification

In a web service based application, changes in QoS values of component web services take place on-the-fly. Therefore, the challenge lies in tracking up-to-date information regarding changes in status of component services and then implementing change reaction decisions as soon as possible.

Many researchers have proposed models and mechanisms for monitoring component web services for dynamic reconfiguration of a CWS [21], but to reconfigure a web service composition in a timely manner when change events happen during its execution is an important issue and still an open question [7,8].

A novel idea to handle web service failures in service oriented software systems uses a push mechanism to notify client application about the failure of a service. A multi agent system tracks a web service on the provider side, and informs the client application as soon as the service fails. This approach is expected to reduce delay in making the replacement decisions. It is also easy on resources as the solution is distributed, and therefore monitoring overhead is also divided between service consumer and provider.

An agent based approach for handling web service availability issues uses the concept of mobile agents, which run at service provider side and provide latest information regarding the web service to the service user. Therefore, service user can arrange alternatives of a failed service even before invoking it. In addition, it also proposes service replacement strategies for optimizing the process execution.

E. Adaption in case of QoS degradation and performance bottleneck

The proposed framework will trigger adaptation as soon as a web service executing in the service composition becomes unavailable or its aggregate QoS values degrade at a provider. This will replace the faulty web service with the next best service (of the same category) available in the local repository. If the last service of a particular category is used, an alert is raised to retrieve more services matching the criteria from the external repository. If the web services alternatives do not exist in the service registry, then the search space is expanded to distant or premium service repositories. Therefore, a potential set of services are always ready for replacement. Lastly, if no matching candidate services are found, then the application may be terminated.

3.3 System Implementation

The prototype of proposed framework is implemented using Java EE. The experiment is conducted on an Intel Core i5 processor with 4 GB RAM running on Windows 7 using Tomcat server and JADE 7 [24, 25]. The web services used in the experiment are described and selected with quality parameters. In order to save time, an abstract service composition is available before execution of the workflow starts.

The process execution starts when a user hits the service client module on a (mobile) device as shown in Figure 1. The service client prompts the user to enter the type of problem (e.g. accident case or a sudden heart attack). The workflow of the service composition starts by finding the current location of user with help of the location locator service. Then an *emergency hospital finder* service finds the appropriate hospitals on the basis of the user location and the specialized services required in the case. The map and time services provide best route related information. If the user had opted for an ambulance service, then a transport management service provider is invoked. User's location and best route from the source to destination is passed as input to the transport service provider.

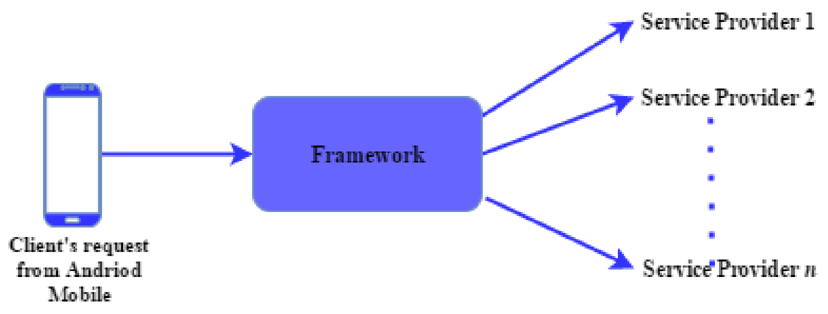

Figure 1. The service composition

4. Results And Analysis

4.1 Performance Evaluation

A performance of a system can be evaluated on the basis of several criteria such as execution time, throughput, and scalability. The execution time is the time spent in servicing a request which also includes time the framework spends in handling the change events i.e. when the change event is detected, and the replacement of a web service is invoked. Throughput is the number of requests completed in unit time.

Figure 2 shows CWS execution time in three different situations 1) a best case that does not need to adapt, 2) a reactive solution to handle change events and, 3) proposed framework. The request was repeated 50 times, and average response time was calculated. In the first case, request is services in 0.28 sec as the system operates in a static environment, and no change event happens. However, such a case is suitable for comparison only to mark a benchmark case. In the real world, there cannot be a web services based solution that does not need to handle change events as the underlying environment is based on the Internet, and is continuously changing. In the second case, the execution time turns out to be 0.65 seconds. While in third case the execution time is around 0.45 seconds. It is remarkably less than the second case that follows a reactive approach to handle change events. The main reason for the time gap in these two cases is that our proposed approach uses a preselected set of candidate services for replacement in case a component web service degrades or becomes a performance bottleneck.

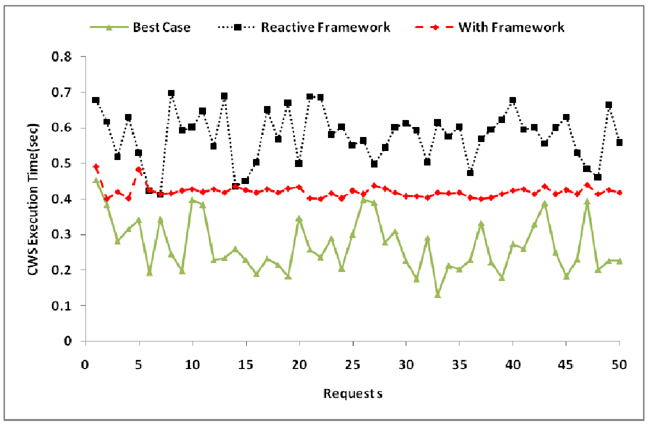

Figure 2. CWS Execution Time (seconds) (1) Best case, (2) Reactive Framework, (3) With Framework

4.2 Analyzing the Quality of Solution

The QoS attributes such as execution time, throughput, availability, and reliability of component web services play a vital role to determine the aggregate QoS of CWS. Figures 3 to 6 show the performance of QoS attributes of CWS performed for simultaneous 100 requests. Figure 3 demonstrate that the execution time of the CWS remains stable even if there is a QoS degradation of component web services or if any web service acts as a performance bottleneck. The average execution time of CWS is nearly 0.45 seconds; though there is a rise in execution time in first ten requests which is warmup time taken by the system to start the workflow. Figure 4 shows that throughput of the service composition improves over period of time. The throughput of proposed system increases as the number of requests is increased over time. After certain time interval the throughput of proposed system becomes stable even if the component web services are creating performance bottleneck for the service composition. Figure 5 and 6 depicts the reliability and availability of the service composition respectively. The results demonstrate that values of reliability and availability are consistently good on an average. The small variations in Figure 5 and 6 are depicted because of the time taken to substitute the component web service which is acting a performance bottleneck for the service composition execution.

Therefore, the proposed framework maintains a stable performance of various QoS attributes in a service composition execution even if the values of QoS attributes degrade over time. The variations in QoS values in a dynamic service execution environment is due to QoS degradation of component web services which should be replaced with better alternatives to improve the quality of solution of a workflow.

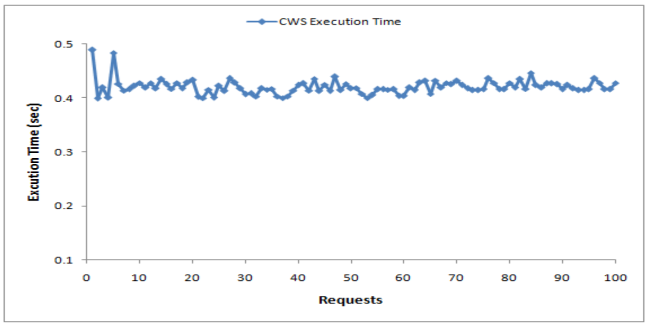

Figure 3. Execution time of the CWS

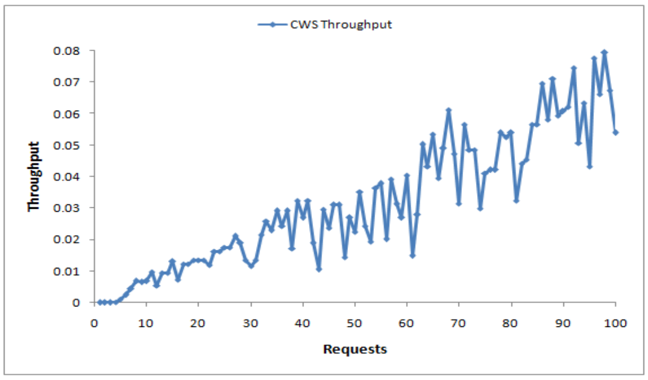

Figure 4. Throughput of the CWS

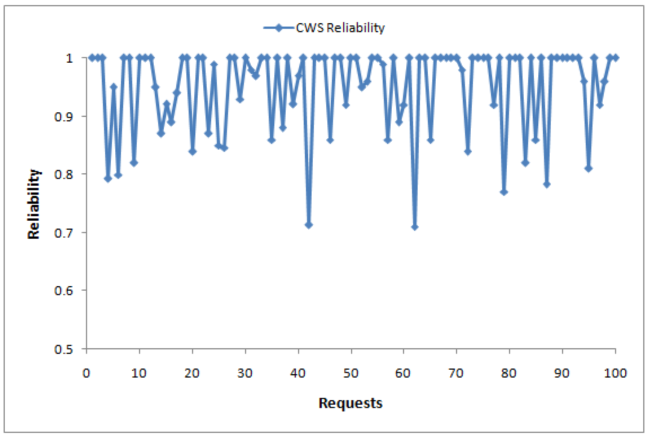

Figure 5. Reliability of the CWS

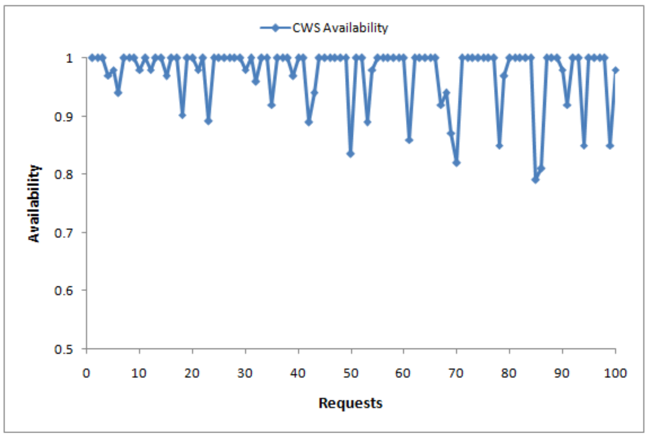

Figure 6. Availability of the CWS

5. CONCLUSIONS

The monitoring and adaptation of web service composition according to the changing scenario at runtime has drawn a lot of attention in the web services based solutions. This paper proposes a distributed monitoring and adaptation framework to keep track of the performance bottleneck of component web services. This paper follows a preventive approach in invocation of component web service with degraded QoS. Additionally, the applicability of the self-adaptive system on the various quality dimensions, such as reliability, availability and throughput is discussed. The adaptation process is flexible enough as it takes into consideration different QoS requirements of different clients.

The experimental results demonstrate that the proposed approach performs better even if component web services are creating performance bottlenecks. The quality of solution is efficient for QoS parameters such as execution time, throughput, reliability and availability. In future, we plan to extend the framework for global optimization of web service composition using self-learning techniques.

REFERENCES

[1] Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., & Pohl, K. (2008). A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, *15*(3), 313-341.

[2] Liang, Q., Lee, B., & Hung, P. (2014). A rule-based approach for availability of service by automated service substitution. *Softw.,Pract. Exper. 44(1)* , 47-76.

[3] Psaier, H., Juszczyk, L., Skopik, F., Schall, D., & Dustdar, S. (2010, September). Runtime behavior monitoring and self-adaptation in service-oriented systems. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (pp. 164-173). Ieee.

[4] Guinea, S., Kecskemeti, G., Marconi, A., &Wetzstein, B. (2011, December). Multi-layered monitoring and adaptation. In *International Conference on Service-Oriented Computing* (pp. 359-373). Springer, Berlin, Heidelberg.

[5] Zeng, L., Lei, H., Jeng, J. J., Chung, J. Y., & Benatallah, B. (2005, July). Policy-driven exception-management for composite web services. In *Seventh IEEE International Conference on E-Commerce Technology (CEC'05)* (pp. 355-363). IEEE.

[6] Friedrich, G., Fugini, M. G., Mussi, E., Pernici, B., &Tagni, G. (2010). Exception handling for repair in service-based processes. *IEEE Transactions on Software Engineering*, *36*(2), 198-215

[7] He, Q., Xie, X., Wang, Y., Ye, D., Chen, F., Jin, H., & Yang, Y. (2017). Localizing Runtime Anomalies in Service-Oriented Systems. *IEEE Transactions on Services Computing*, *10*(1), 94-106.

[8] Campos, G. M., Souto Rosa, N., & Ferreira Pires, L. (2017, January). Adaptive service composition based on runtime verification of formal properties. In *Proceedings of the 50th Hawaii International Conference on System Sciences*.

[9] Cheng, S. P., Lu, X. M., & Zhou, X. Z. (2014). Globally optimal selection of web composite services based on univariate marginal distribution algorithm. Neural Computing and Applications, 24(1), 27-36. https://doi.org/10.1007/s00521-013-1440-9

[10] Chen, Y., Huang, J., Lin, C., & Hu, J. (2015). A partial selection methodology for efficient qos-aware service composition. IEEE Transactions on Services Computing, 8(3), 384-397.

[11] Cremene, M., Suciu, M., Pallez, D., & Dumitrescu, D. (2016). Comparative analysis of multi-objective evolutionary algorithms for QoS-aware web service composition. *Applied Soft Computing*, *39*, 124-139.

[12] Liu, X., Shi, W., Kale, A., Ding, C., & Yu, Q. (2017). Statistical Learning of Domain-Specific Quality-of-Service Features from User Reviews. *ACM Transactions on Internet Technology (TOIT)*, *17*(2), 22.

[13] Moustafa and ZangMoustafa, A., & Zhang, M. (2012, June). Towards Proactive Web Service Adaptation. In *CAiSE*(pp. 473-485).

[14] Angarita, R., Cardinale, Y., &Rukoz, M. (2014). Reliable composite web services execution: towards a dynamic recovery decision. Electronic Notes in Theoretical Computer Science, 302, 5-28.

[15] Xiong X., Wang P., Zhang Q., Pu C. L. (2014). Revisiting proactive service-oriented architecture: from design and implementation to validation and performance improvement, International Journal of Services ComputingVol. 2, No. 1, pp. 1-12.

[16] Chahal, K. K., Kahlon, N. K., &Narang, S. B. (2017). Improving the QoS of a Composite Web Service by Pruning its Weak Partners. In *Requirements Engineering for Service and Cloud Computing* (pp. 271-290). Springer International Publishing.

[17] Zhu, J., He, P., Zheng, Z., & Lyu, M. R. (2017). Online QoS prediction for runtime service adaptation via adaptive matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, *28*(10), 2911-2924.

[18] El Hog, C., Djemaa, R. B., & Amous, I. (2014). A User-Aware Approach to Provide Adaptive Web Services. *J. UCS*, *20*(7), 944-963.

[19] Nabli, H., Cherif, S., Djmeaa, R. B., & Amor, I. A. B. (2018, May). SADICO: Self-ADaptIve Approach to the Web Service COmposition. In *International Conference on Intelligent Interactive Multimedia Systems and Services* (pp. 254-267). Springer, Cham.

[20] Angarita, R. (2015, July). Responsible objects: Towards self-healing internet of things applications. In *2015 IEEE International Conference on Autonomic Computing* (pp. 307-312). IEEE.

[21] Angarita, R., Rukoz, M., &Cardinale, Y. (2016). Modeling dynamic recovery strategy for composite web services execution. World Wide Web, 19(1), 89-109.

[22] Yu, C. H. (1977). Exploratory data analysis. *Methods*, *2*, 131-160.

[23] Fakhfakh, N., Verjus, H., Pourraz, F., &Moreaux, P. (2013). QoS aggregation for service orchestrations based on workflow pattern rules and MCDM method: evaluation at design time and runtime. *Service Oriented Computing and Applications*, *7*(1), 15-31.

[24] Bellifemine, F., Caire, G., Poggi, A., &Rimassa, G. (2008). JADE: A software framework for developing multi-agent applications. Lessons learned. *Information and Software Technology*, *50*(1), 10-21.

[25] JADE mobile agent platform, http://jade.tilab.com

[26] Wang, H., Wang, X., Hu, X., Zhang, X., & Gu, M. (2016). A multi-agent reinforcement learning approach to dynamic service composition. *Information Sciences*, *363*, 96-119.

[27] Gavvala, S. K., Jatoth, C., Gangadharan, G. R., & Buyya, R. (2019). QoS-aware cloud service composition using eagle strategy. *Future Generation Computer Systems*, *90*, 273-290.

AUTHORS

Navinderjit Kaur Kahlon received Ph.D. degree and Master's degree in Computer Science and Engineering from Guru Nanak Dev University, India. She is currently working as an Assistant Professor in the Department of Computer Science, Guru Nanak Dev University, India. The research interests include Service Oriented Computing, Distributed Systems, Mobile Agents and Web Technologies.

Kuljit Kaur Chahal received the Ph.D. in Computer Science in 2011. She is working as an Associate Professor in the Department of Computer Science, Guru Nanak Dev University, India. Her research interests are Distributed Computing, Web Services Security, and Open Source Softwares.