

Handling Faults in Service Oriented Computing: A Comprehensive Study

Roaa ElGhondakly, Sherin Moussa(✉), and Nagwa Badr

Information Systems Department, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt
{roaa.ahmed, sherinmoussa, nagwabadr}@cis.asu.edu.eg

Abstract. Recently, service-oriented computing paradigms have become a trending development direction, in which software systems are built using a set of loosely coupled services distributed over multiple locations through a service-oriented architecture. Such systems encounter different challenges, as integration, performance, reliability, availability, etc., which made all associated testing activities to be another major challenge to avoid their faults and system failures. Services are considered the substantial element in service-oriented computing. Thus, the quality of services and the service dependability in a web service composition have become essential to manage faults within these software systems. Many studies addressed web service faults from diverse perspectives. In this paper, a comprehensive study is conducted to investigate the different perspectives to manipulate web service faults, including fault tolerance, fault injection, fault prediction and fault localization. An extensive comparison is provided, highlighting the main research gaps, challenges and limitations of each perspective for web services. An analytical discussion is then followed to suggest future research directions that can be adopted to face such obstacles by improving fault handling capabilities for an efficient testing in service-oriented computing systems.

Keywords: Fault tolerance · Fault prediction · Fault injection · Quality of Service · Service testing · Service oriented computing

1 Introduction

Service-oriented systems are built through combining a set of distributed services that are previously developed for reuse. These loosely coupled services communicate with each other through a predefined protocol, forming a service composition to achieve a complex task [1, 2]. However, the performance and quality of the individual services might be higher than those of the resultant service composition, due to some integration issues, unavailability of services, dependability between services, etc., leading to faults occurrence. Four main mechanisms are investigated to achieve service dependability facing service compositions with respect to the quality, performance and reliability: fault avoidance, fault removal, fault prediction and fault tolerance. Fault avoidance methods minimize the occurrence of faults, whereas fault removal methods aim to remove faults as soon as possible. Fault prediction techniques predict the number of faults before

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12252, pp. 947–959, 2020.
https://doi.org/10.1007/978-3-030-58811-3_67

occurrence, while fault tolerance detects faults and tries to solve them early from the system [3]. In this paper, a comprehensive study is conducted to investigate the studies handling web service faults with respect to different perspectives as shown in Fig. 1. A thorough analytical evaluation is discussed for the challenges and limitations encountered in each perspective in the state of art, revealing the main research gaps. In addition, a set of future directions are proposed for powerful testing in service-oriented computing (SOC) systems. The rest of the paper is organized as follows: Sect. 2 discusses the main research studies that tackle faults in service-oriented systems according to the given hierarchy. Section 3 analyzes the revealed challenges and limitations, whereas Sect. 4 proposes various future directions to consider maximizing testing benefits in service-orientation. Finally, Sect. 5 concludes the proposed study.

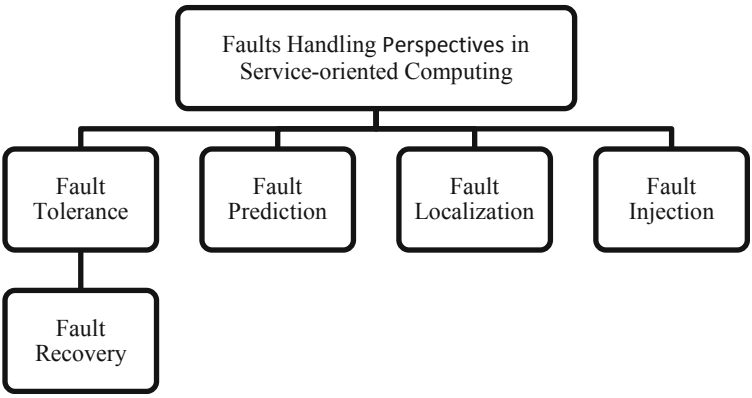

Fig. 1. The perspectives of faults handling in service-oriented computing (SOC)

2 Faults Manipulation in Service-Oriented Computing

Many studies have addressed diverse perspectives of faults handling in Service-oriented computing (SOC) systems, as fault tolerance, fault prediction, fault injection, etc. Following the hierarchy in Fig. 1, a comprehensive discussion is presented to investigate each perspective of faults handling in the following sub-sections.

2.1 Fault Tolerance

Fault tolerance allows a system to keep on operating while one or more of its components stop working or tend to fail [4]. Thus, fault tolerance should consist of a detection method and a recovery method. Most of the proposed approaches in this perspective required to combine other fault mechanisms in order to achieve fault tolerance, like fault localization, fault injection and fault recovery. Gupta et al. [5], considered fault tolerance from the web service composition point of view, taking into consideration some quality of service (QoS) attributes. A fault injection mechanism was used to inject two types of faults: network faults, by examining the unavailability

of a service, and logical faults by observing incorrect values. Authors then tried to detect their existence and recover from them, without trying to find the already existed faults. In addition, only two types of faults were considered, ignoring any other types of faults.

In [6], the authors studied two existing fault tolerance mechanisms: Time redundancy and space redundancy, with their three strategies: Retry, Active replication and Passive replication. In Retry strategy, if a service fails, it would retry and run frequently to either work correctly or reach the limit of trials. Active and Passive replications depend on using replicas to overcome failures. In Active strategy, it invokes all services and the first returned suitable one is used, whereas in Passive strategy, the primary service is invoked first, then if it fails, a backup replication is used instead. They proposed integrated simulation algorithms combining all three strategies to control services and modify the ones that did not meet the desired requirements in the form of time, space, etc. However, although the simulation has given high reliability, they did not consider reliability after applying actual data. Fekih et al. [7] used a sensor-based method to detect and find faults' location, and then used two repair algorithms to correct faults before system crash; single service reconfiguration (SSR) and multiple service reconfiguration (MSR) with a metaheuristic search algorithm. However, they did not evaluate the efficiency of their results within comparison with other approaches. Authors in [8] used two ranking algorithms HITS based Service Component Ranking Algorithm (HSCR) and PageRank based Service Component Ranking Algorithm (PSCR) to rank services, where top services could be used later since their fault tolerance ability would be high. However, they encountered critical drawbacks, as they did not consider neither fault detection nor localization methods, whereas fault tolerance should include detection and recovery methods. In addition, localization methods help in finding the location of faulty service, which eases the recovery process. They only proposed the optimum services set to use instead of the faulty ones, in which only simple services were investigated neglecting QoS attributes.

A fault recovery approach was utilized in [9] through an error handling procedure, but it suffered from its weak response time that affected the overall performance. Chen et al. [10] proposed a 3-phase mechanism, in which these phases were invocation, synchronization and exception with the aid of petri nets to show the behavior of services. The invocation phase selected the services for composition, the synchronization phase obtained matching services for the selected ones, while the exception phase excluded the services that may cause faults as well as its connected ones. This mechanism was inefficient when services' resources increase, affecting the reliability of the proposed approach. Authors in [11] proposed a fault tolerance approach for mobile ad hoc networks based on a checkpointing technique, in which the faulty services were recovered by substituting them with the ones that satisfied QoS constraints using a fuzzy-rated system. However, their evaluation was based on simulation rather than real world problems. Besides, they encountered a single point of failure issue that would affect the reliability and availability of their approach.

Another rejuvenation fault tolerance scheme was introduced in [12], consisting of 3 sub phases: failure detection, age evaluation of components, and checkpointing method for rejuvenation. However, the results lacked accuracy as a fault detection metric, as well as maintaining availability. Miltiadis et al. in [13] proposed an approach for

predicting the average execution time for faulty applications counting on a checkpoint mechanism. Although this approach improved the performance of faulty software, it did not propose a recovery method to overcome the existing faults. Authors in [14] investigated the impact of checkpointing interval selection on the cloud application performance, as well as finding the relation between the checkpointing interval and the failure probability. Yet, this study did not consider any fault recovery techniques. In [15], a fault injection method was used to inject faults in an application. It detected such faults using a logging mechanism in order to assess the performance of the applied fault tolerance mechanism, without discussing how faults were handled or recovered. Moreover, Jhawar and piuri in [16] analyzed the characteristics and reasons for failure of 2 fault types: crash and byzantine faults, as well as their corresponding fault tolerance mechanisms. The studies in [17] and [18] have considered fault tolerance approaches only, discarding other fault handling perspectives, in which one study addressed fault tolerance with load balancing algorithms, while the other addressed checkpointing approaches. In [19], Angarita et al. proposed a dynamic fault tolerance approach that could select the superior recovery strategy to correct faults based on analyzing some QoS parameters, the execution state and environment state. However, in practice, depending on the execution state and environment state criteria might not be possible [20]. Xu et al. in [21] discussed the impact of applying a fault recovery mechanism on time and cost, while addressing the problem of workflow scheduling in cloud computing via developing a heuristic-based algorithm. Nevertheless, they ignored that tasks might also fail, not only components.

Thus, most of the related studies applied fault detection and recovery to fulfill fault tolerance process, while ignored by few of them. Some have taken into consideration the QoS for its great impact on fault existence. Moreover, some studies applied their proposed approaches on simulated data rather than real data, which affected the truthfulness of their results.

2.2 Fault Prediction

Fault prediction is the process of predicting whether a software is faulty or not, as well as the number of faults that may appear in a software [22]. There exist only few studies that investigated fault prediction in the service-oriented computing paradigm. In [23], the authors proposed a framework to study the ability of source code metrics to predict faults in web services. They managed to predict whether a web service is faulty by applying 5 feature selection techniques to select such metrics. In addition, they considered 6 machine learning algorithms to predict faults: Naïve Bayes, Artificial Networks (ANN), Adaptive Boosting (AdaBoost), decision tree, Random Forests and Support Vector Machine (SVM). However, the main drawback was that they could not predict the number of faults. Ding et al. in [24] predicted the service composition reliability and located the faulty services using a spectrum-fault-localization technique, but the proposed method was not able to predict the number of faults as well. Authors in [25, 26] investigated some machine learning techniques for fault prediction, as classification algorithms and multi-layer perceptron, by developing a web service on Azure cloud platform to predict faulty services but neglected predicting the number of faults. Chatterjee et al. [27] proposed a prediction algorithm through integrating

clustering and fuzzy algorithms to predict more than one fault at a time within one run, whereas in [28], a clustering method was proposed to detect faults using k-means++. A new set of metrics was introduced in [29] to make the fault detection process more efficient as well as to increase its performance, but they only validated their metrics on PHP based projects. However, the different approaches proposed in [27, 28] and [29] focused on predicting faulty components but neglected to predict the type and number of faults.

Consequently, most of the previous studies focused on predicting faults in service-oriented computing, ignoring the idea of predicting the number of faults, which would help in deciding how faulty the component is. In addition, it would allow controlling faults to increase services reliability and performance.

2.3 Fault Localization

Fault localization is the process of locating the faulty code or component [30, 31]. Despite of the existence of many software fault localization techniques, there were only few studies directed to fault localization in SOC paradigms [32]. Sun et al. in [33] proposed a fault localization approach for WS-BPEL programs, where switching and slicing methods were used. This study addressed fault localization for the interaction section of WS-BPEL programs only, neglecting the other sections. An automatic modeling mechanism was proposed in [34] using the "Policy View" of each service to identify fault location by building a belief network per service, but they ignored the types of located faults. Some studies have combined fault localization techniques to other fault handling perspectives, as with fault tolerance in [7] and with fault prediction in [24]. These combinations improved the reliability and availability of services, where locating their faults facilitated their correction timely and effectively.

Hence, there is still an eager need to investigate this perspective, with the aim of increasing web services reliability and performance. This includes considering different types of located faults, investigating different fault localization techniques, and defining the optimum techniques to apply in SOC systems. In addition, combining fault localization techniques with more fault handling perspectives is an encouraging trend that would elevate high performance and efficient fault handling.

2.4 Fault Injection

Fault injection is the process of injecting faults to the application to test its reliability and performance by trying to detect and remove such faults [35]. Qian et al. in [36] presented a fault injection process that counted on the place where faults can be injected and what type of faults can be injected. However, only sub-service stability coverage criterion was considered, in which more coverage criteria were highly required to evaluate the efficiency and reliability of the proposed approach. In [37], an approach was proposed using fault injection techniques to assist the failure diagnosis when the same failure takes place. In addition, it used fault localization to detect the root cause of the failure and to recover the system through fixing such faults. Some limitations were encountered, as the proposed approach could neither diagnose all kinds of failures, nor support multiple distributed systems paradigms. Another

combined method was proposed in [38] to estimate the dependability of distributed systems. Contract-based and model-based cyber physical systems (CPS) were combined, applying a fault injection method to check the system dependability of the resulting model.

Thus, many studies have used fault injection methods for multiple reasons, as to evaluate and test a system [36, 38], to generate representative failure data [39], to check performance and reliability [36, 37], to assess dependability [40] and for security management [41, 42]. Furthermore, fault injection techniques were combined with other fault handling perspectives, as fault tolerance in [5] to increase the reliability and performance of the system under test. This indicates that there is still a growing interest on how and where to inject faults. A closer investigation is expected to study fault injection mechanisms with respect to SOC paradigms, considering its distributed nature and interoperability.

3 Challenges and Limitations of Faults Handling in SOC

Despite of having software faults as a comprehensive research field that has diverse contributions, handling faults in SOC is still insufficient. This is due to the different nature of web services rather than regular software systems, in which extra testing is highly demanded, and faults can occur easily for various reasons. In accordance with the related state of art, most of the previous studies considered fault tolerance as the main perspective for faults handling in service-oriented computing paradigms. Some combined it with other perspectives to boost the reliability of the system under test. A serious lack is witnessed in the research efforts proposed for the other fault handling perspectives. This raises many challenges and limitations as follows:

a) Fault tolerance methods-related Challenges
   - Detecting many types of faults and their count is a persistent challenge, where the proposed studies did not cover multiple fault types, neglecting the number of faults. They only determined whether there exists a fault or not by injecting faults and trying to detect/localize them, ignoring the detection of actual faults rather than synthetic ones.
   - Recovery strategies and fault prediction techniques require vital advancement with respect to the fault localization studies, in which only synthetic fault injection processes were considered, which lack to represent the diversity of faults that SOC systems can face. This would increase the performance and reliability of such systems.

b) Fault prediction methods-related Challenges
   - The need for language-independent approaches, in which some studies proposed approaches for specific types of applications written in certain languages only, as WS-BPEL, PHP projects, etc.

- Semantic faults need to be considered, where most of the current studies detected faults that occur due to integration issues between components, unavailability of one of the composition services, etc., while ignoring faults that occur due to a semantic issue between services.

c) Fault injection and localization methods-related Challenges

- Handling faults in SOC systems should consider faults leading to task failure, not only faults leading to service failure. This would require new fault localization and injection approaches to address this different level of granularity.
- Investigating how and where to inject faults to improve the performance.

d) Generic Challenges

- Most of the current approaches showed high efficiency using simulated services, whereas in practice, these approaches could be inapplicable for real world applications.
- Addressing different evaluation measurements, metrics and coverage criteria to ensure the quality of the system under test.
- Consider other fault handling perspectives as fault prevention.
- The proposed approaches need to be more generic and applicable for different distributed systems.

Table 1 summarizes all deduced drawbacks and challenges at the main fault handling studies in the SOC paradigm, associated with the corresponding fault handling perspective considered, applied mechanisms, measured evaluation criteria, and the datasets used to examine the applied mechanisms.

Table 1. Deduced drawbacks and challenges in fault handling studies in SOC

Ref#	Fault handling perspective	Mechanism	Drawbacks	Challenges	Dataset	Evaluation
[5]	Fault tolerance and injection	Detect logic and network faults	Only two types of faults were considered	Detect the type and number of faults	Supply chain management system	Time complexity
[6]	Fault tolerance	Time and space redundancy with retry, active and passive replication	Reliability based on simulated data only	Consider actual faults	Composite web service of bank transaction	Reliability, round-trip time, failure rate
[7]	Fault tolerance	Single and multiple service reconfiguration	Considered only checkpointing techniques for faults detection	Fault detection techniques	QWS dataset [43]	Response time

(*continued*)

Table 1. (*continued*)

Ref#	Fault handling perspective	Mechanism	Drawbacks	Challenges	Dataset	Evaluation
[8]	Fault tolerance	HITS based and PageRank based service component ranking algorithms	Neglected QoS attributes, did not detect the faulty service	Consider more metrics	A real-world dataset	Failure rate
[9]	Fault tolerance	Selection of services, provided top-k compositions, fault recovery	Response time affected by similarity detection methods	Performance issues	OWLS-TC2 Semantic Web Services Ver 1.1	Response time, Efficiency, failure rate
[10]	Fault tolerance	Invocation, synchronization and exception with petri nets	Efficiency issues when service resources increase	Efficiency challenges	QWS dataset [43]	Run time, Reliability
[11]	Fault tolerance	Checkpointing technique then recovery via a fuzzy rated system	Considered checkpointing techniques only for fault detection	Fault detection techniques	Simulation of services	Service composition Efficiency, Failure rate, Recovery time
[12]	Fault tolerance	Rejuvenation fault tolerance scheme with detection, evaluation and checkpointing	Accuracy of fault detection	Performance measurements	A private cloud application	Accuracy and availability
[13]	Fault tolerance	Predict the average execution time of a program with presence of failure using checkpointing	Considered checkpointing only to improve the execution time and performance	Fault detection and recovery techniques	Multiple software applications with different loop sizes	Execution time
[14]	Fault tolerance	Investigate via simulation the impact of checkpointing interval selection on the performance of a SaaS cloud	Considered checkpointing to investigate the relation bet. the checkpointing interval and failure probability	Fault detection and recovery techniques	Simulated application	Performance metrics
[15]	Fault tolerance and injection	Inject fault and measure performance of fault tolerance	Did not state the used fault tolerance mechanism and how it recovered faults	Fault tolerance mechanisms	Some web services applications	Response time

(*continued*)

Table 1. (*continued*)

Ref#	Fault handling perspective	Mechanism	Drawbacks	Challenges	Dataset	Evaluation
[19]	Fault tolerance and recovery	Fault tolerance model to determine the best recovery strategy	Depended on metrics inapplicable for real world applications	Considering actual data	Generated 1000 CWSs	Efficiency and Effectiveness
[21]	Fault tolerance and recovery	Study the impact of fault recovery mechanism on time and cost	Did not discuss the optimum recovery strategy	Recovery strategy challenges	Real workflows	Cost and time
[23]	Fault prediction	Feature selection and machine learning algorithms	Stated whether a component is faulty or not	Detect type and number of faults	WSDL files from Multiple datasets	ROC, f-measure, Accuracy, Precision, RECALL
[24]	Fault prediction and localization	Spectrum-fault-localization technique	Did not state the type and number of faults	Detect the type and number of faults	An online shop example	Effectiveness and reliability
[26]	Fault prediction	Machine learning and classification algorithms	Did not state the type and number of faults	Detect the type and number of faults	Drupal, moodle datasets	Average fault prediction rate
[27]	Fault prediction	Integrate clustering and fuzzy algorithms	Focused on fault existence	Predict the type and number of existing faults	HTTP logs of certain web pages	Dynamic and dunn validity indices
[28]	Fault prediction	A clustering method for detecting faults using k-means ++	Detected faults only	Predict the type and number of existing faults	4 different datasets	Performance measurements and defect rate
[29]	Fault prediction	Applied a new set of metrics for efficient fault detection process	Detected faulty services only	Predict the type and number of faults, Tasks failure	Datasets from 3 open source web applications	Performance measurements
[34]	Fault localization	Automatic modeling mechanism using “Policy View” of each service	Consume more memory	Performance and quality limitations	Multiple topologies	Performance, System overhead and Viability
[36]	Fault injection	Analyzed multiple fault injection techniques	Needed to consider more coverage criteria	Coverage criteria limitation	Experiment subjects: ATM, NextDate	Performance, Response time

(*continued*)

Table 1. (*continued*)

Ref#	Fault handling perspective	Mechanism	Drawbacks	Challenges	Dataset	Evaluation
[37]	Fault injection and localization	Assisted failure diagnosis and applied fault localization and recovery	cannot diagnose all kinds of failures, cannot support multiple distributed systems	Generalization issue	OpenStack cloud platform	Variation, Accuracy

4 Suggested Future Directions in Faults Handling for SOC

New fault handling approaches in SOC systems should address the diversity of fault sources with respect to the interoperability and loose coupling of services, which generate emergent types of faults special to the nature of SOC systems. Future research should consider innovative approaches to anticipate the number of faults in a tested system, as well as discovering more fault types, as functionality faults, syntactic faults, control flow faults, semantic faults, service dependency in service compositions faults, etc. In addition, fault localization approaches should investigate hybrid methods by combining static, dynamic and execution slicing based methods and program spectrum-based methods, etc. [35].

Moreover, system recovery after finding faults is a very important process to consider. Therefore, new recovery strategies should be addressed that fit the distributed architecture, increased federation, expanded intrinsic interoperability and vendor diversification of the SOC paradigm. Besides, new fault handling perspectives, like fault prevention/avoidance techniques, should be tackled in order to improve the performance and quality of SOC systems. In addition, merging multi-fault perspectives together should be investigated to increase the dependability and reliability of the tested system. Consequently, fault handling in SOC is still an emerging field of research that expects more investigations to follow.

5 Conclusion

Even though software faults occur frequently, in case of service-oriented computing (SOC) systems, faults are more likely to exist due to the nature of web services, in which the availability of some services is not guaranteed. In this study, we thoroughly investigated the main fault handling approaches that have been presented with respect to the SOC paradigms from different perspectives, including fault tolerance, fault prediction, fault localization, fault recovery and fault injection. A detailed comparison is conducted to analyze the challenges and limitations of the current fault handling approaches directed to the SOC systems. Moreover, a set of suggested future directions are proposed for efficient fault handling capabilities in SOC systems, in which current research efforts are still insufficient.

References

1. Mikalsen, T., Wohlstadter, E., Desai, N., Rouvellou, I., Tai, S.: Transaction policies for service-oriented computing. Data Knowl. Eng. **51**(1), 59–79 (2004)
2. Rao, J., Su, X.: A survey of automated web service composition methods. In: Cardoso, J., Sheth, A. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30581-1_5
3. Griffiths, N., Chao, K.-M. (eds.): Agent-Based Service-Oriented Computing. AIKP. Springer, London (2010). https://doi.org/10.1007/978-1-84996-041-0
4. Agarwal, H., Sharma, A.: A comprehensive survey of fault tolerance techniques in cloud computing. In: 2015 International Conference on Computing and Network Communications (CoCoNet). IEEE (2015)
5. Gupta, R., Kamal, R., Suman, U.: A QoS-supported approach using fault detection and tolerance for achieving reliability in dynamic orchestration of web services. Int. J. Inf. Technol. **10**(1), 71–81 (2017). https://doi.org/10.1007/s41870-017-0066-z
6. Shu, Y., Wu, Z., Liu, H., Gao, Y.: A simulation-based reliability analysis approach of the fault-tolerant web services. In: 2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), pp. 125–129. IEEE (2016)
7. Fekih, H., Mtibaa, S., Bouamama, S.: The dynamic reconfiguration approach for fault-tolerance web service composition based on multi-level VCSOP. Procedia Comput. Sci. **159**, 1527–1536 (2019)
8. Chen, L., Liu, L., Shang, J.: Fault tolerance for web service based on component importance in service networks. In: Proceedings of the Fifth International Conference on Network, Communication and Computing (2016)
9. Kargar, A., Emadi, S.: Fault tolerance in automatic semantic web service composition based on QoS-awareness using BTSC-DFS algorithm. In: 5th International Conference on Web Research (ICWR), pp. 50–54. IEEE (2019)
10. Chen, L., Fan, G., Liu, Y.: A formal method to model and analyse QoS-aware fault tolerant service composition. Int. J. Comput. Sci. Eng. **12**(2–3), 133–145 (2016)
11. Veeresh, P., Sam, R.P., Bin, C.S.: Reliable fault tolerance system for service composition in mobile Ad Hoc network. Int. J. Electr. Comput. Eng. **9**, 2523–2533 (2019)
12. Liu, J., Zhou, J., Buyya, R.: Software rejuvenation based fault tolerance scheme for cloud applications. In: 2015 IEEE 8th International Conference on Cloud Computing. IEEE (2015)
13. Siavvas, M., Gelenbe, E.: Optimum checkpoints for programs with loops. Simul. Model. Pract. Theory **97**, 101951 (2019). https://doi.org/10.1016/j.simpat.2019.101951. ISSN 1569-190X
14. Stavrinides, G.L., Karatza, H.D.: The impact of checkpointing interval selection on the scheduling performance of real-time fine-grained parallel applications in SaaS clouds under various failure probabilities. Concurrency Comput. Pract. Exp. **30**(12), e4288 (2018)
15. Farj, K., Smeda, A.: A methodology for evaluating fault tolerance in web service applications. In: Proceedings of the 15th International Conference on Applied Computer Science (ACS 2015), pp. 188–191 (2015)
16. Jhawar, R., Piuri, V.: Fault tolerance and resilience in cloud computing environments. In: Computer and Information Security Handbook, 1 January 2017, pp. 165–181. Morgan Kaufmann, Burlington (2017)
17. Kumar, S., Rana, D.S., Dimri, S.C.: Fault tolerance and load balancing algorithm in cloud computing: A survey. Int. J. Adv. Res. Comput. Commun. Eng. **4**(7), 92–96 (2015)
18. Vargas-Santiago, M., Hernández, S.E., Rosales, L.A., Kacem, H.H.: Survey on web services fault tolerance approaches based on checkpointing mechanisms. JSW **12**(7), 507–525 (2017)

19. Angarita, R., Rukoz, M., Cardinale, Y.: Modeling dynamic recovery strategy for composite web services execution. World Wide Web **19**(1), 89–109 (2015). https://doi.org/10.1007/s11280-015-0329-1
20. Bashari, M., Bagheri, E., Du, W.: Self-adaptation of service compositions through product line reconfiguration. J. Syst. Softw. **144**, 84–105 (2018)
21. Xu, H., Yang, B., Qi, W., Ahene, E.: A multi-objective optimization approach to workflow scheduling in clouds considering fault recovery. KSII Trans. Internet Inf. Syst. (2016)
22. Rathore, S.S., Kumar, S.: A study on software fault prediction techniques. Artif. Intell. Rev. **51**(2), 255–327 (2017). https://doi.org/10.1007/s10462-017-9563-5
23. Bhandari, G.P., Gupta, R., Upadhyay, S.K.: An approach for fault prediction in SOA-based systems using machine learning techniques. Data Technol. Appl. **53**(4), 397–421 (2019)
24. Ding, Z., Xu, T., Ye, T., Zhou, Y.: Online prediction and improvement of reliability for service oriented systems. IEEE Trans. Reliab. **65**(3), 1133–1148 (2016)
25. Malhotra, R.: A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. **27**, 504–518 (2015)
26. Catal, C., Akbulut, A., Ekenoglu, E., Alemdaroglu, M.: Development of a software vulnerability prediction web service based on artificial neural networks. In: Kang, U., Lim, E.-P., Yu, J.X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10526, pp. 59–67. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67274-8_6
27. Chatterjee, S., Roy, A.: Novel algorithms for web software fault prediction. Qual. Reliab. Eng. Int. **31**(8), 1517–1535 (2015)
28. Öztürk, M.M., Cavusoglu, U., Zengin, A.: A novel defect prediction method for web pages using k-means++. Exp. Syst. Appl. **42**(19), 6496–6506 (2015)
29. Biçer, M.S., Diri, B.: Predicting defect prone modules in web applications. In: Dregvaite, G., Damasevicius, R. (eds.) ICIST 2015. CCIS, vol. 538, pp. 577–591. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24770-0_49
30. Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F.: A survey on software fault localization. IEEE Trans. Softw. Eng. **42**(8), 707–740 (2016)
31. Zou, D., Liang, J., Xiong, Y., Ernst, M.D., Zhang, L.: An empirical study of fault localization families and their combinations. IEEE Trans. Softw. Eng. (2019)
32. Ghawate, S.B., Shinde, S.: Survey of software fault localization for web application. Int. J. Curr. Eng. Technol. **5**(3), 1525–1529 (2015)
33. Sun, C.A., Ran, Y., Zheng, C., Liu, H., Towey, D., Zhang, X.: Fault localisation for WS-BPEL programs based on predicate switching and program slicing. J. Syst. Softw. **135**, 191–204 (2018)
34. Tang, Y., Cheng, G., Xu, Z., Chen, F., Elmansor, K., Wu, Y.: Automatic belief network modeling via policy inference for SDN fault localization. J. Internet Serv. Appl. **7**(1), 1–13 (2016). https://doi.org/10.1186/s13174-016-0043-y
35. Wong, W.E., Debroy, V.: A survey of software fault localization. Department of Computer Science, University of Texas at Dallas (2009)
36. Qian, J., Wu, H., Chen, H., Li, C., Li, W.: Fault injection for performance testing of composite web services. Int. J. Performability Eng. **14**(6), 1314–1323 (2018)
37. Pham, C., et al.: Failure diagnosis for distributed systems using targeted fault injection. IEEE Trans. Parallel Distrib. Syst. **28**(2), 503–516 (2016)
38. Dal Lago, L., Ferrante, O., Passerone, R., Ferrari, A.: Dependability assessment of SOA-based CPS with contracts and model-based fault injection. IEEE Trans. Ind. Inf. **14**(1), 360–369 (2017)
39. Irrera, I., Vieira, M.: Towards assessing representativeness of fault injection-generated failure data for online failure prediction. In: 2015 IEEE International Conference on Dependable Systems and Networks Workshops. IEEE (2015)

40. Herscheid, L., Richter, D., Polze, A.: Experimental assessment of cloud software dependability using fault injection. In: Camarinha-Matos, L.M., Baldissera, T.A., Di Orio, G., Marques, F. (eds.) DoCEIS 2015. IAICT, vol. 450, pp. 121–128. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16766-4_13
41. Salas, M.I., De Geus, P.L., Martins, E.: Security testing methodology for evaluation of web services robustness-case: XML injection. In: 2015 IEEE World Congress on Services. IEEE (2015)
42. Bhor, R.V., Khanuja, H.K.: Analysis of web application security mechanism and attack detection using vulnerability injection technique. In: 2016 International Conference on Computing Communication Control and automation (ICCUBEA). IEEE (2016)
43. Yin, Y., Li, Y.: Towards dynamic reconfiguration for QoS consistent services based applications. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 771–778. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34321-6_61