Review Article

Overview on Fault Tolerance Strategies of Composite Service in Service Computing

Junna Zhang, Ao Zhou , Qibo Sun, Shangguang Wang , and Fangchun Yang

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Correspondence should be addressed to Ao Zhou; hellozhouao@gmail.com

Received 11 April 2018; Accepted 23 May 2018; Published 19 June 2018

Academic Editor: Fuhong Lin

Copyright © 2018 Junna Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In order to build highly reliable composite service via Service Oriented Architecture (SOA) in the Mobile Fog Computing environment, various fault tolerance strategies have been widely studied and got notable achievements. In this paper, we provide a comprehensive overview of key fault tolerance strategies. Firstly, fault tolerance strategies are categorized into static and dynamic fault tolerance according to the phase of their adoption. Secondly, we review various static fault tolerance strategies. Then, dynamic fault tolerance implementation mechanisms are analyzed. Finally, main challenges confronted by fault tolerance for composite service are reviewed.

1. Introduction

With the rapid advance of SOA, there are greater numbers of self-contained, self-describing, loosely coupled, and modular component services in the Internet. To implement sophisticated business applications, one or more services are combined into value-added and coarse-grained service oriented system, that is, composite service. Nowadays, a growing number of enterprises employ composite services to shorten the software development cycle, reduce development costs, and ultimately implement their business processes [1].

However, faults are prone to happen during the execution of composite service. That is because a large proportion of component services are deployed in the best-effort and unreliable Internet, especially in the Mobile Fog Computing environment. Mobile Fog Computing is put forward to enable computing directly at the edge of the network, which can deliver new services for the future of the Internet. However, there are many resource-poor devices in the Mobile Fog Computing environment, for example, routers, switches, and base stations. Composite services are more prone to fault if component services are deployed on resource-poor devices [2]. Therefore, fault tolerant strategy has become a crucial necessity for building reliable composite service. In recent years, many scholars and organizations have engaged in fault tolerant strategies research and put forward various fault

tolerant strategies. In this paper, an overview of key fault tolerant strategy for composite service is presented.

We categorize the fault tolerant strategies according to the phase of their adoption. When fault tolerance strategy is employed in the design phase of composite service, it is referred to as a static fault tolerant strategy. When it is adopted during the execution phase, the strategy is referred to as a dynamic fault tolerant strategy [3]. There are various implementation schemes for static and dynamic fault tolerance strategies, so an overview of main literature about them is presented in this paper.

The rest of this paper is organized as follows. The next section presents the category of fault tolerance. Static fault tolerance strategies are analyzed in Section 3. Dynamic fault tolerance strategies are discussed in Section 4. Brief conclusion about the challenge of fault tolerance strategies is given in Section 5. The last section concludes the paper.

2. Category for Fault Tolerance Strategy

To enhance the reliability and trustworthiness of composite service, various fault tolerance strategies have been put forward. The major fault tolerance strategies can be divided into static and dynamic fault tolerance strategy via the phase of their adoption. Static fault tolerance strategy is employed in the design phase of composite service, and it is usually

FIGURE 1: Fault tolerant composite service design and execution. The rectangle module presents the module of traditional composite service design and execution. The grey rectangle module presents the module that implements the fault tolerance function. Lined arrow and dashed arrow represent the data flow between the modules.

to implement the fault tolerance requirements of the user. Moreover, the designer considers the fault that is possible to occur during the execution stage and implements the coping strategy in the design stage. Dynamic fault tolerance strategy is usually adopted when the composite service really fails, and its purpose is to troubleshoot and resume execution of the composite service.

In order to make the category easier to understand, fault tolerance modules are inserted into traditional composite service design and execution modules [4]. All modules are illustrated in Figure 1.

In the design stage of the composite service, the composite service developers need to analyze fault tolerant requirements besides the functional requirements to implement complex tasks of the consumer. According to the results of the static fault tolerant requirements (which are obtained from the static fault tolerance requirements analysis module), the developer can select an appropriate strategy (which is obtained from the static fault tolerance selection module) and employ it in the service selection process. There are various traditional static fault tolerant strategies, for example, the high-certainty, high-trustworthiness, and high-reliability component services selection, exception handling and transaction techniques combination, and component services ranking. Besides, there is a kind of special fault during the execution of composite service, which can be referred to as Byzantine fault. To handle Byzantine fault, Byzantine fault

tolerance strategy must be performed at the design time. All aforementioned static fault tolerance strategies will be analyzed in Section 3.

A fault may occur during the run-time of composite service. Therefore, the execution states of composite service should be collected by the run-time monitoring module. When a fault occurs, fault tolerant requirements are firstly analyzed by the dynamic fault tolerance requirements analysis module according to the fault state. Then an appropriate fault tolerant strategy is selected via the dynamic fault tolerance strategy selection module. Forward recovery, backward recovery, and checkpoint are main dynamic fault tolerance strategies. Finally, fault tolerance strategy recovers the execution of composite service from the fault state. All aforementioned dynamic fault tolerance strategies will be discussed in Section 4.

3. Static Fault Tolerance Strategies

To construct a reliable and trustworthy composite service, static fault tolerant strategies are adopted at the stage of design. The purpose of static fault tolerance strategy is to select reliable and trustworthy component service for composite service. Static fault tolerance strategies are usually carried out during the service selection phase [5]. There are various static fault tolerant strategies, for example, the high-certainty component selection [6], high-trustworthiness

component selection [7], high-reliability component selection [8, 9], fault tolerance based on exception handling and transaction techniques [10], and component services ranking [11].

The above-mentioned strategies can only handle traditional fault of composite service, but they cannot handle Byzantine fault. A Byzantine fault poses a serious threat to the composite service via sending conflicting information to other component services. To mask this type of fault, Byzantine fault tolerance strategy must be adopted [12]. Hence, researchers keep exploring and working on this study.

3.1. Traditional Static Fault Tolerance Strategies. Besides functional requirements, nonfunctional requirements (or QoS constraints, e.g., total execution time should be less than 10 s) should be satisfied in a composite service design. However, component service providers only provide the average QoS values or even incorrect values to improve utilization, which would lead to the violation of QoS constraints. That is to say, there will be a fault. To avoid this situation, component service with high certainty and high reputation should be chosen in selection phase [13, 14].

To select the component services with the highest certainty for composite service, a reliable and efficient approach is put forward in [6]. Firstly, the approach adopts the probability theory and information theory to filter component services with low certainty. Then a reliable fitness function is devised via using 0-1 integer programming. Finally, the component services with the highest certainty are selected based on the fitness function.

According to the collaboration reputation, a service selection approach is proposed in [7] to select the trustworthy component service. The collaboration reputation is constructed on a component service collaboration network that includes two metrics. One metric is invoking reputation, which can be calculated via the recommendation of other component services. The other metric is invoked reputation, which can be calculated according to the interaction frequency among component services. Finally, a trustworthy component service selection algorithm is put forward based on collaboration reputation.

To improve the fault tolerance of the composite service, a novel service selection approach is proposed in [15]. The approach consists of two decision phases. In the first decision phase, the finding of reliable component service is defined as a multiple criteria decision-making problem. And a decision model is constructed to address this problem. In the second decision phase, service selection problem is formulated as an optimization problem based on QoS requirements, and a convex hull approach is presented to solve this optimization problem.

In [10], a fault tolerant framework that is referred to as FACTS is proposed for composite service. To design a fault tolerant mechanism that combines exception handling and transaction techniques, this paper identifies a set of high level exception handling strategies and presents a new taxonomy of transactional component services. Moreover, two modules (a specification module and a verification module) are also designed for assisting service designers

in constructing fault handling logic conveniently and correctly.

Component service ranking is another approach for fault tolerance. In [11], FTCloud, a component service ranking framework, is put forward. Firstly, the framework employs two ranking algorithms. The first algorithm adopts invocation structures and frequencies of component service to make significant component ranking. The other ranking algorithm recognizes the significant component services from all composite services by fusing the system structure information and the designer's wisdom of application. After the component service ranking phase, a selection algorithm for optimal fault tolerance strategy is proposed, which can automatically supply optimal fault tolerance strategy for the significant components.

Traditional static fault tolerant strategies are usually employed in the design phase of composite service, so the key research issue of them is not the execution time reduction but the accuracy improvement [16]. Meanwhile, for aforementioned strategies that are only adopted in the design phase, their effectiveness during the execution is another key research issue. To our knowledge, there are few strategies that consider both accuracy and effectiveness during the execution.

3.2. Byzantine Fault Tolerance Strategies. During the execution of composite service, a failed component service may send conflicting information to another component service, which constitutes various threats to the consistency of composite service. This type of fault is known as Byzantine fault [17]. To mask Byzantine fault during the execution phase, the composite service must employ a fault tolerance strategy in the design phase [18]. In recent years, some scholars engage in studying Byzantine fault tolerance strategy.

To tolerate Byzantine faults of composite service, a framework, BFT-WS, is designed and used in [19, 20]. Firstly, BFT-WS adopts the standard technology of composite service (i.e., SOAP) to construct Byzantine fault tolerance service. Employing standard technology can ensure the interoperability of component services. BFT-WS is designed as a pluggable module. Therefore, the implementation of BFT-WS needs minimum change to the composite service. Finally, the key fault tolerance schemes employed in BFT-WS are designed based on the notable Castro and Liskov's Byzantine fault tolerance approach.

A practical algorithm, Perpetual, is proposed in [21]. Perpetual can tolerate Byzantine faults of deterministic n-tier composite service. Interaction between services with different number of replica is allowed in Perpetual. In addition, Perpetual supports not only long-running active threads of computation but also asynchronous invocation and processing. Therefore, Perpetual can improve performance and flexibility over other protocols.

To make the coordination of Web Services Business Activities (WS-BA) more trustworthy, a lightweight Byzantine fault tolerance algorithm is put forward in [22]. Depending on careful study of the threats of the WS-BA coordination services and comprehensive analysis of the state model, the algorithm is lightweight designed. In order to implement

FIGURE 2: Dynamic fault tolerance strategies. During the execution of composite service, the fault tolerance strategy type can be selected according to the fault state and fault tolerance requirements when a fault occurs.

Byzantine fault tolerance and state machine replication of the WS-BA coordination services, the algorithm uses source ordering rather than total ordering.

To orchestrate delivery of reliable composite services, a hybrid asynchronous Byzantine fault tolerant protocol, GEMINI, is proposed in [23]. Firstly, GEMINI decomposes composite services' abstract workflows from its implementation because it sustains dynamic components provisioning. Then, GEMINI guarantees the reliability of service delivery modules via a lightweight Byzantine fault tolerant protocol. Moreover, GEMINI invokes multiple component services concurrently to realize component service redundancy. Finally, GEMINI employs a single leader Byzantine faults tolerance technology to optimize the current Byzantine fault tolerant protocol.

To handle Byzantine fault, group communication is obligatory among the component service replicas. However, if the traffic between different replicas of component service is very heavy, the response time of a component service may remarkable increase. That is because component services are usually distributed on the Internet. So a key research issue of the Byzantine fault tolerant is reducing the response time of component service. Meanwhile, component service replicas are usually provided by different service providers. Therefore, how to guarantee seamless communication between replicas is another key research issue.

4. Dynamic Fault Tolerance Strategies

A component service may fail during the execution of composite service. The fault must be repaired via dynamic fault tolerance strategies; otherwise, it will lead to the failure of composite service. The current dynamic fault tolerance strategies include forward recovery, backward recovery, and checkpoint, which are illustrated in Figure 2. To ensure the

whole composite service in a consistent state even suffering from fault, it is necessary to provide component services with transactional property (all or nothing (every component service of composite service must either be executed successfully or have no effect whatsoever)). Backward recovery and forward recovery are two basic fault tolerance strategies supported by component service's transactional properties. If the faulty component service can be retried [24], replicated [25], or substituted [26], forward recovery is allowed. If the effects produced by the faulty component service need to be compensated [27], backward recovery is allowed [28]. However, users need to wait a long time to get the desired response when forward recovery is adopted, and users are unable to get the desired answer to their queries when backward recovery is adopted [29]. Taking checkpoint is another dynamic fault tolerance strategy. Current execution state and partial results are taken as a snapshot, which is returned to the user when a fault occurs. The checkpointed composite service can be restarted from the latest saved state, and the aggregated transactional attributes are not affected [28]. The recent researches of the dynamic fault tolerance are discussed in the following sections.

Different dynamic fault tolerance strategies need to be adopted for the different faults that occur during the execution of the composite service, and some scholars have specifically studied dynamic fault tolerance strategies selection [7]. Therefore, the main literature about it is presented in Section 4.4.

4.1. Forward Recovery. For forward recovery, the composite service tries to fix the fault without stopping execution. Retry, replication, and substitution can be used for forward recovery [30].

A solution based on forward recovery is proposed in [31] to provide reliable composite service. The solution has

no impact on the autonomy of the component services while exploiting their possible support for fault tolerance. The key issue of this solution is to construct cooperative atomic actions that have a well-defined behavior. Firstly, the notion of Web Service Composition Action (WSCA) is defined according to the concept of coordinated atomic action. Then dependable actions are structured by WSCA, and fault tolerance can be gotten as an emergent property of aggregation of several potentially nondependable services [32].

Fault can be repaired by the substitution. A substitution policy is proposed in [33], which substitutes a subset of component services (includes failed component service) with another equivalent subset. When a fault occurs, all subsets containing the failed component service are identified. Then the subsets that are equivalent to the failed one are determined. Finally, the equivalent subsets are ranked, and the failed subset is substituted by the best equivalent subset.

Replication creates redundant component services (replicas) for composite service. When a request from the user is assigned to all replicas, the technology is called active replication. Otherwise, only one replica acts as the primary one that responds to the request, and the backup replica takes over only after the primary one fails. The technology is called passive replication [34].

WS-Replication, a framework for seamless replication of composite services, is proposed in [35]. To increase the service availability, the framework permits the deployment of a component service in a set of sites. One of the stand-out features of WS-Replication is that replication is done concerning component service autonomy and only SOAP is used to interact across sites. What is more, WS-Multicast (one of the major components of WS-Replication) can also be used as a self-governed component for reliable multicast in a component service setting [36].

In [37], a distributed replication strategy evaluation and selection framework for fault tolerant composite service is proposed. Based on the proposed framework, various replication strategies are compared by using the theoretical formula and experimental results. Moreover, a strategy selection algorithm based on both objective performance information and subjective requirements of users is proposed.

Each of the aforementioned strategies has its own advantages and disadvantages and is employed for specific fault tolerance scenarios. The composite service developer should first analyze the requirements of the user and the possible fault scenario and then select appropriate strategy [38].

4.2. Backward Recovery. When a fault occurs, backward recovery should be adopted if the effects need be compensated [39].

Some scholars employed exception handling strategies to realize the backward recovery. For example, Liu et al. [10] present a framework named FACTS for fault tolerance of transactional composite service. FACTS combines exception handling and transaction techniques to improve fault tolerance of composite services. Firstly, the framework identifies a set of high level exception handling strategies. Then, a

specification module is designed to help service designers to construct correct fault-handling logic. Finally, a module is devised to automatically implement fault-handling logic in WS-BPEL.

An efficient framework for fault tolerance of transactional composite service is proposed in [40]. For recovery from fault, the framework realizes a backward recovery method based on unfolding processes of Coloured Petri-Nets. The framework can be realized in distributed/shared memory system.

According to the transactional properties of component service, a framework, called FaCETa, is proposed in [41]. FaCETa employs service replacement and Coloured Petri-Nets' unrolling processes to tolerate fault. Besides, experimental results show that FaCETa efficiently realizes fault tolerant strategies for the transactional composite service with small overhead.

An approach that dynamically calculates the composite service's reliability to improve the performance of backward recovery is proposed in [42]. Firstly, a model of reliability is presented according to the doubly stochastic model and renewal processes. Then, to help the calculation of complex composite services, a bounded set strategy is briefly presented. Finally, a fault tolerance model is constructed via backward recovery block techniques.

Guillaume et al. [39] focus on checking the correctness of compensation via invariant preservation. Therefore, a correct-by-construction approach, which uses the Event-B algorithm to deal with runtime compensation, is put forward based on refinement and proof. The approach can be used as a foundational module for the compensation of run-time composite service. Meanwhile, a formal model is defined for equivalent, degraded, and upgraded service compensations.

Backward recovery needs to go back to a consistent state to repair the fault correctly. Therefore, a key issue of it is how to save the execution state of the composite service. In addition, how to look for an alternative execution path from the consistent state is another key issue of backward recovery.

4.3. Checkpoint. Checkpoint refers to execution states of composite service gathered by orchestration in a certain time, and the composite service can return to a previous specific state for fault tolerance [43].

Marzouk et al. [44] propose a flexible approach for composite service's execution. The approach synchronizes all flow branches of the composite service. Then a recovery state that permits saving a consistent checkpoint is constructed. When a fault or a QoS violation occurs, the failed process or a subset of running instance may be migrated to another server and restarted according to the checkpoint image.

The traditional "all-or-nothing" is too restrictive for composite service. Checkpoint techniques can relax the atomicity based on the transactional properties of component service. Based on checkpoint and transactional properties, a model that measures the fuzzy atomicity of composite service is presented in [45]. "All-or-nothing" attribute is relaxed into a fuzzy "all-something-or-almost-nothing" attribute.

Based on Coloured Petri-Nets, a checkpoint approach is proposed in [46]. If a fault occurs, the approach relaxes

the all-or-nothing attribute by executing a transactional composite Web service as much as possible and taking a snapshot of faulted state. In other words, the approach returns partial answers to the user as soon as possible. According to the snapshot, the user can resume the composite service without dropping the work previously done.

In [29], the unfolding processes of the Coloured Petri-Nets that control the execution of a transactional composite Web service are checkpointed if a fault occurs. In such way, users can first get partial responses as soon as they are obtained, and the composite service can be restarted from an advanced point of execution.

4.4. Dynamic Fault Tolerance Strategy Selection. Different types of faults may happen during the execution of the composite service. Therefore, different fault tolerance strategies should be employed to recover them [47]. There are some literatures that study how to select the most appropriate fault tolerance strategy [48].

The fault tolerance strategy selection has a significant effect on the QoS of composite service [49]. Therefore, Zheng et al. [50] investigated the problem of selecting an optimal fault tolerance strategy for building reliable composite services. They formulated the user's requirements as local constraints and global constraints and modelled the fault tolerance strategy selection as an optimization problem. A heuristic algorithm is presented to efficiently solve the optimization problem.

In [51], a QoS-aware fault tolerant middleware is proposed to make the dependability of composite service. The middleware includes a user-collaborated QoS model, a set of fault tolerance strategies, and a context-aware algorithm that (dynamically and automatically) determines the optimal fault tolerance strategy for both stateful and stateless composite services.

To maintain the required QoS even in the presence of fault, a novel approach is proposed in [4]. This approach builds on the top of the execution system of composite service and carries out the QoS monitoring. The result of QoS monitoring determines the selection of the fault tolerance strategy in case of fault.

To select appropriate fault tolerance strategy, Shu et al. [52] considered that the reliability of composite services must be analyzed. They proposed a tree-based composition structure model called the Fault-Tolerant Composite Web Service Tree (FCWS-T). Firstly, nodes in FCWS-T are separated into two types, which are control nodes and service nodes. Then, a reliable simulation method is put forward based on FCWA-T, and it can efficiently analyze the reliability of a complex composite service. Finally, an appropriate fault tolerance strategy is selected according to the reliability.

Using priority selector and fault handler, an approach of fault tolerance for service oriented architecture is put forward in [53]. Firstly, the approach selects the first priority level scheme quickly when a fault has been detected. If the fault cannot be handled, the second priority level scheme is selected by a fault handler for average performance. Otherwise, the lowest priority level scheme is employed to handle the fault.

5. Discussion and Open Challenges

Fault tolerance strategy has achieved great development in recent decades and has been successfully applied for solving various faults during the execution of composite service. However, due to the special structure (i.e., based on SOA) and complex and unreliable execution environment of composite service, there are numerous challenges in the research of fault tolerance strategy.

(1) Regarding a compatible development platform for component service, to construct fault tolerant composite service, various fault tolerance strategies should be employed. Most strategies try to choose another component service to replace the faulty one. However, component services are usually developed by different organizations based on different development platform, which leads to some differences between them. These differences have negative impact on the effectiveness of fault tolerance strategy. But few literatures consider this issue now. The difference would not be eliminated unless there is a compatible development platform for component service. Therefore, one of the future researches of fault tolerance strategy is to develop a compatible development platform for component service.

(2) For effectiveness validation of fault tolerance strategy in a real network environment, in recent years, plenty of fault tolerant strategies are proposed for different fault of composite service. However, most of their effectiveness is only validated in a simulation environment. However, real network environment is complex and changeable, and all existing simulation platforms cannot simulate it. Therefore, how to validate the effectiveness of existing fault tolerance strategy in a real network environment needs further study.

6. Conclusion

Building a highly reliable composite service has become a key issue with the prevalence of component services in the Internet. Therefore, many fault tolerance strategies are proposed in recent years. In this paper, fault tolerance strategies are divided into static and dynamic fault tolerance strategies. For implementation of static fault tolerance strategy, there are the high-certainty, high-trustworthiness, and high-reliability component services selection, fault tolerant mechanism of combined exception handling and transaction techniques, and component services ranking. Besides, Byzantine fault tolerant strategy can mask a special kind of fault, that is, Byzantine fault. The overview of the main literature about them is discussed. For implementation of dynamic fault tolerance strategy, there are forward recovery, backward recovery, and checkpoint. The overview of main literature about them is analyzed. Moreover, some challenges in the research of fault tolerance strategy are also provided.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61602054, 61472047, and 61571066) and Beijing Natural Science Foundation (no. 4174100).

References

- [1] Y. Ma, S. Wang, P. C. Hung, C. H. Hsu, Q. Sun, and F. Yang, "A highly accurate prediction algorithm for unknown web service QoS value," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 511–523, 2016.
- [2] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the Workshop on Mobile Big Data (Mobidata '15)*, pp. 37–42, ACM, Hangzhou, China, June 2015.
- [3] A. A. von Davier, "Service fault tolerance for highly reliable service-oriented systems: an overview," *Science China Information Sciences*, vol. 58, no. 1, pp. 1–12, 2015.
- [4] A. Immonen and D. Pakkala, "A survey of methods and approaches for reliable dynamic service compositions," *Service Oriented Computing and Applications*, vol. 8, no. 2, pp. 129–158, 2014.
- [5] N. Aljeri, K. Abrougui, M. Almulla, and A. Boukerche, "A reliable quality of service aware fault tolerant gateway discovery protocol for vehicular networks," *Wireless Communications and Mobile Computing*, vol. 15, no. 10, pp. 1485–1495, 2015.
- [6] S. Wang, L. Huang, L. Sun, C.-H. Hsu, and F. Yang, "Efficient and reliable service selection for heterogeneous distributed software systems," *Future Generation Computer Systems*, vol. 74, pp. 158–167, 2016.
- [7] S. Wang, L. Huang, C.-H. Hsu, and F. Yang, "Collaboration reputation for trustworthy Web service selection in social networks," *Journal of Computer and System Sciences*, vol. 82, no. 1, part B, pp. 130–143, 2016.
- [8] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 65, no. 8, pp. 2348–2362, 2016.
- [9] A. Zhou, S. Wang, Z. Zheng, C.-H. Hsu, M. R. Lyu, and F. Yang, "On cloud service reliability enhancement with optimal resource usage," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 452–466, 2016.
- [10] A. Liu, Q. Li, L. Huang, and M. Xiao, "FACTS: A framework for fault-tolerant composition of transactional web services," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 46–59, 2010.
- [11] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, 2012.
- [12] L. Chen and W. Zhou, "Byzantine Fault Tolerance with Window Mechanism for Replicated Services," in *Proceedings of the 2015 Fifth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, pp. 1255–1258, Qinhuangdao, China, September 2015.
- [13] S. Wang, A. Zhou, W. Lei, Z. Yu, C.-H. Hsu, and F. Yang, "Enhanced user context-aware reputation measurement of multimedia service," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 12, no. 4s, 2016.
- [14] S. Wang, Z. Zheng, Z. Wu, M. R. Lyu, and F. Yang, "Reputation measurement and malicious feedback rating prevention in web service recommendation systems," *IEEE Transactions on Services Computing*, vol. 8, no. 5, pp. 755–767, 2015.
- [15] W. Wang, Z. Huang, and L. Wang, "ISAT: An intelligent Web service selection approach for improving reliability via two-phase decisions," *Information Sciences*, vol. 433–434, pp. 255–273, 2018.
- [16] A. Zhou, S. Wang, B. Cheng et al., "Cloud Service Reliability Enhancement via Virtual Machine Placement Optimization," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 902–913, 2017.
- [17] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.
- [18] S. Murugan and B. Muthukumar, "Detection and Elimination of Byzantine Faults Using SOAP Handlers in Web Environment," *Modern Applied Science (MAS)*, vol. 9, no. 9, 2015.
- [19] W. Zhao, "BFT-WS: A Byzantine Fault Tolerance Framework for Web Services," in *Proceedings of the 2007 11th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOC Workshops)*, pp. 89–96, Annapolis, MD, USA, October 2007.
- [20] W. Zhao, "Design and implementation of a Byzantine fault tolerance framework for Web services," *The Journal of Systems and Software*, vol. 82, no. 6, pp. 1004–1015, 2009.
- [21] S. L. Pallemulle, H. D. Thorvaldsson, and K. J. Goldman, "Byzantine fault-tolerant Web services for n-tier and service oriented architectures," in *Proceedings of the 28th International Conference on Distributed Computing Systems, ICDCS 2008*, pp. 260–268, chn, July 2008.
- [22] H. Chai, H. Zhang, W. Zhao, M.-S. Michael, and L. E. Moser, "Toward trustworthy coordination of web services business activities," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 276–288, 2013.
- [23] I. Elgedawy, "GEMINI: A Hybrid Byzantine Fault Tolerant Protocol for Reliable Composite Web Services Orchestrated Delivery," *International Journal of Computer Theory and Engineering*, vol. 8, no. 5, pp. 355–361, 2016.
- [24] A. Erradi, P. Maheshwari, and V. Tosic, "Recovery policies for enhancing Web services reliability," in *Proceedings of the ICWS 2006: 2006 IEEE International Conference on Web Services*, pp. 189–196, usa, September 2006.
- [25] M. Vargas-Santiago, L. Morales-Rosales, S. Pomares-Hernandez, and K. Drira, "Autonomic Web Services Enhanced by Asynchronous Checkpointing," *IEEE Access*, 2017.
- [26] K. Ali, D. Tarun, and D. Mohemmed, "Computation of QoS While Composing Web Services," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 3, 2017.
- [27] S. Wang, T. Lei, L. Zhang, C.-H. Hsu, and F. Yang, "Offloading mobile data traffic for QoS-aware service provision in vehicular cyber-physical systems," *Future Generation Computer Systems*, vol. 61, pp. 118–127, 2016.
- [28] R. Angarita, M. Rukoz, and Y. Cardinale, "Modeling dynamic recovery strategy for composite web services execution," *World Wide Web*, vol. 19, no. 1, pp. 89–109, 2016.
- [29] M. Rukoz, Y. Cardinale, and R. Angarita, "FaCETa*: Checkpointing for transactional composite web service execution based on petri-Nets," in *Proceedings of the 3rd International Conference on Ambient Systems, Networks and Technologies, ANT 2012 and 9th International Conference on Mobile Web*

- Information Systems, MobiWIS 2012*, pp. 874–879, can, August 2012.
- [30] N. Zhang, J. Wang, Y. Ma, K. He, Z. Li, and X. Liu, “Web service discovery based on goal-oriented query expansion,” *The Journal of Systems and Software*, vol. 142, pp. 73–91, 2018.
 - [31] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy, “Coordinated forward error recovery for composite Web services,” in *Proceedings of the 22nd International Symposium on Reliable Distributed Systems, SRDS 2003*, pp. 167–176, ita, October 2003.
 - [32] A. Zhou, S. Wang, C.-H. Hsu, M. H. Kim, and K.-S. Wong, “Virtual machine placement with (m, n)-fault tolerance in cloud data center,” *Cluster Computing*, pp. 1–13, 2017.
 - [33] S. Gupta and P. Bhanodia, “A fault tolerant mechanism for composition of Web services using subset replacement,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 8, pp. 3080–3085, 2013.
 - [34] M. Vargas-Santiago, S. E. Pomares-Hernandez, L. A. Morales Rosales, and H. Hadj-Kacem, “Survey on Web Services Fault Tolerance Approaches Based on Checkpointing Mechanisms,” *Journal of Software*, pp. 1–19, 2017.
 - [35] J. Salas, F. Perez-Sorrosal, M. Patiño-Martínez, and R. Jiménez-Peris, “WS-replication: a framework for highly available web services,” in *Proceedings of the 15th International Conference on World Wide Web*, pp. 357–366, Edinburgh, UK, May 2006.
 - [36] A. Zhou, Y. Li, and J. Li, “Efficient Request Assignment Algorithm in Mobile Cloud Computing environment,” *International journal of Web and Grid Service*, pp. 1–17.
 - [37] Z. Zheng and M. R. Lyu, “A distributed replication strategy evaluation and selection framework for fault tolerant Web services,” in *Proceedings of the IEEE International Conference on Web Services, ICWS 2008*, pp. 145–152, chn, September 2008.
 - [38] S. Wang, A. Zhou, C.-H. Hsu, X. Xiao, and F. Yang, “Provision of Data-Intensive Services Through Energy-and QoS-Aware Virtual Machine Placement in National Cloud Data Centers,” *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 290–300, 2016.
 - [39] G. Babin, Y. Aït-Ameur, and M. Pantel, “Web Service Compensation at Runtime: Formal Modeling and Verification Using the Event-B Refinement and Proof Based Formal Method,” *IEEE Transactions on Services Computing*, vol. 10, no. 1, pp. 107–120, 2017.
 - [40] Y. Cardinale and M. Rukoz, “A framework for reliable execution of Transactional Composite Web Services,” in *Proceedings of the International Conference on Management of Emergent Digital EcoSystems, MEDES’11*, pp. 129–136, usa, November 2011.
 - [41] R. Angarita, Y. Cardinale, and M. Rukoz, “FaCETa: Backward and Forward Recovery for Execution of Transactional Composite WS,” in *The Semantic Web: ESWC 2012 Satellite Events*, vol. 7540 of *Lecture Notes in Computer Science*, pp. 343–357, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
 - [42] H. E. Mansour and T. Dillon, “Dependability and rollback recovery for composite web services,” *IEEE Transactions on Services Computing*, vol. 4, no. 4, pp. 328–339, 2011.
 - [43] LY. Chiu, S. Fan, Y. Liu, and M. Mei, “Providing a fault tolerant system in a loosely-coupled cluster environment using application checkpoints and logs,” in *and Mei M. Providing a fault tolerant system in a loosely-coupled cluster environment using application checkpoints and logs*, p. B2, United States Patent: US, 2015.
 - [44] S. Marzouk, A. J. Maâlej, and M. Jmaiel, “Aspect-Oriented Checkpointing Approach of Composed Web Services,” in *Proceedings of the International Conference on Web Engineering (ICWE’10)*, vol. 6385, pp. 301–312, 2010.
 - [45] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz, “Measuring Fuzzy Atomicity for Composite Service Execution,” in *Proceedings of the 2016 2nd International Conference on Open and Big Data (OBD)*, pp. 62–71, Vienna, Austria, August 2016.
 - [46] Y. Cardinale, M. Rukoz, and R. Angarita, “Modeling Snapshot of Composite WS Execution by Colored Petri Nets,” in *Resource Discovery*, vol. 8194 of *Lecture Notes in Computer Science*, pp. 23–44, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
 - [47] R. Gupta, R. Kamal, and U. Suman, “A QoS-supported approach using fault detection and tolerance for achieving reliability in dynamic orchestration of web services,” *International Journal of Information Technology*, vol. 10, no. 1, pp. 71–81, 2018.
 - [48] Y. Liu, Y. Fan, K. Huang, and W. Tan, “Failure analysis and tolerance strategies in web service ecosystems,” *Concurrency Computation*, vol. 27, no. 5, pp. 1355–1374, 2015.
 - [49] J. Wang, P. Gao, Y. Ma, K. He, and P. C. K. Hung, “A Web Service Discovery Approach Based on Common Topic Groups Extraction,” *IEEE Access*, vol. 5, pp. 10193–10208, 2017.
 - [50] Z. Zheng and M. R. Lyu, “Selecting an optimal fault tolerance strategy for reliable service-oriented systems with local and global constraints,” *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 64, no. 1, pp. 219–232, 2015.
 - [51] Z. Zheng and MR. Lyu, “A QoS-aware fault tolerant middleware for dependable service composition,” in *Proceedings of the In Proceedings of the IEEE/IFIP International Conference on Dependable Systems Networks (DSN’09)*, pp. 239–248, 2009.
 - [52] Y. Shu, D. Zuo, H. Liu, Q. Z. Sheng, W. E. Zhang, and J. Yang, “A Tree-Based Reliability Analysis for Fault-Tolerant Web Services Composition,” in *Service Oriented Computing and Applications*, vol. 10601 of *Lecture Notes in Computer Science*, pp. 481–489, Springer International Publishing, Cham, 2017.
 - [53] G. Prasad Bhandari and . Ratneshwer, “Fault Repairing Strategy Selector for ServiceOriented Architecture,” *International Journal of Modern Education and Computer Science*, vol. 9, no. 6, pp. 32–39, 2017.