I.J. Modern Education and Computer Science, 2017, 6, 32-39
Published Online June 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2017.06.05

Fault Repairing Strategy Selector for Service-Oriented Architecture

Guru Prasad Bhandari
DST-CIMS, Banaras Hindu University, Varanasi, 221005, India
Email: guru.bhandari@gmail.com

Ratneshwer
Department of Computer Science, MMV, BHU, Varanasi, 221005, India
Email: ratnesh@bhu.ac.in

Abstract—The success of a service oriented computing significantly depends on its reliability and availability. To achieve better reliability and availability, any fault of the service oriented computing has to be properly handled. Low performance on handling the service fault becomes the most proliferated challenge on fault handling approaches. In this paper, a priority based fault handling strategy selection approach for fault recovery of SOA (Service Oriented Architecture) is proposed using priority selector and fault handler. This approach starts from fault detection method and select the first priority level strategies promptly and if fault could not be handled by first highest level then fault handler selects the second level or intermediate priority level for average performance. As a final in the worst case, least level strategies are applied to resolve the faulty situation. Through experiment, the correctness of the proposed algorithm and the efficiency of the approach are proved. This approach results better performance and high rate of fault repairing.

Index Terms—Fault handling, Fault repairing, Reliability, Logging, Replication.

I. INTRODUCTION

Service Oriented Architecture (SOA) is a popular design paradigm provides architectural style for distributed system to enable applications to be built using service as a key element. Service Oriented Computing (SOC) employs services to support rapid, cost effective development of distributed applications in heterogeneous environments. Interoperability, self-adaptability, dynamic bound are the highly demanded emerging features of SOA but these features are also highly susceptible to service fault. Service fault may be generated by high adaptability and complexity of SOC that eventually may result SOA-based system failure any stage of SOA life cycle.

The objective of this study is to present fault handling or repairing strategies with strategy selector associating with nature of fault. Researchers have focused on general software recovery actions even for the SOA-based fault handling in the same way. Since SOA lies in heterogeneous system, some standalone approaches may not be proper solution for the service-oriented fault handling approach. Degree of heterogeneity makes fault tolerance approach highly desirable and difficult to achieve. Some participating components of the SOA-based system may be owned by several organizations thus backward recovery (rollback) techniques to handle fault would not be suitable due to various authentication and authorization access control to different services that belong to different organizations. We have studied various type of SOA-based faults and categorized these and modelled them on priority basis according to their suitability for the whole recovery performance improvement.

From our literature review, we found reliability and performance are the highly concentrated challenges. Besides these two major challenges, other challenges like adaptation, interoperability, scalability, security, management have also obtained great concern of the researchers. Service composition is a great feature but most challenging feature to realize Service Oriented Computing (SOC). Researchers were highlighting on fault analyzing approaches and techniques of SOA same like distributed system till the end of 2012 but later onward machine learning approaches used as optimizing the reliability and performance on fault handling techniques.

La et al. [1] have defined a fault as a problem that occurs when a service invocation made by a service based system results in some abnormality at runtime. The fault analysis takes fault data as input and determines a suitable remedial strategy for the fault instance [2]. A service may be healthy, impacted or faulty at any stage of SOA life cycle.

Rest of this paper is organized as section II discusses related work. Section III focuses on commonly preferred fault handling strategies of the service-oriented computing (SOC). Overview of our proposed approach with brief description of strategy selector among the collection of strategies according to the fault situation is presented in section IV. Performance analysis is calculated in section V. Finally, section VI summarizes our work with the conclusion.

II. Related Work

Some relevant related works are presented in this section with the recent state-of-art on fault handling of service-oriented system. From our literature review, we found that local recovery and backward recovery are the commonly used recovery strategies. Local recovery tries to fix the fault in current state of error in a way that is similar to compensation using exception handling approaches. Backward recovery assists to restore the system to a previous stable version where the fault was not occurred at all.

Some of the mechanisms are implemented as WS-BPEL plug-ins. Baresi et al. [3] have used WSCol and WSRel to improve the performance on fault handling. WSRel is used for backward recovery to restore the service state with the previous stable state. They have developed event handler, fault handler and compensation handler. Event handler responds two types of events; message events and alarms based on timers. Fault handler catches faults in local scope, tries to handle it with suitable strategies if not possible then the fault is propagated to the enclosing scope. Finally, compensation handler works for backward recovery in fault case and applies the transactional constraint and initiates the handler programmatically.

BPEL does not provide any recovery strategies for SOA faults unless predefined by developers at design time. Friedrich et al. [4] defines reparability as "An activity Ai is repairable iff, after Ai has been executed, it is possible to execute a sequence of repair". Ruan et al. [5] have developed TEHL (Task Level Exception Handling Language) that reduces abnormal events that interrupts the normal execution of workflows. TEHL implements exception handling strategies like delay, repeat etc. Wang et al. [2] have concerned about integrated handling of business constraint violations and runtime environment faults for dynamic service composition. SOA specific fault can be categorized as publishing fault, discovering fault, composition fault, binding fault and execution fault according the SOA life cycle stages as proposed by [6]. Service unavailability fault [7] [8], byzantine fault [9] [10], prescribed policy violation [11], timeout exception [12], SLA (Service Level Agreement) claim fault [13], latent errors and dormant faults[14], adaptation faults [15], interaction faults [16], network traffic [17] of service-oriented computing faults are noted to be highly emphasized faults by the researchers.

Detecting a faulty service is very difficult task. The fault can only be detected in execution step when the service is actually executed. Several fault detection techniques are proposed by the researchers like impact analysis [7], dependency discovery [18], set-covering algorithm [19] [19] [20] [21] [22], fuzzy reasoning based diagnosis algorithm [23] [23] [24], process structure analyzing [4], timed-automata [25], logging as traces collection mechanism [26] [27] [28] [29], pattern-based technique [30], event-based approach [31], etc.

Several researchers have emphasized on mainly two recovery techniques; internal recovery and external recovery. Ermagan et al. [32] have proposed fault tolerance approach based on architectural pattern using interactions among components and interception/routing mechanism. Jensen et al. [33] have proposed fault propagation approach for service composition. It ensures the flexibility and robustness in fault handling. They have mainly categories fault recovery into three techniques; internal recovery, forward recovery and third backward recovery. The proposed FaultHandler service takes appropriate measures. Performance analysis is not calculated on their study.

III. Service Oriented Architecture

Service oriented architecture is a popular design paradigm for distributed system. According to IBM definition [34]- "SOA is a set of architectural principles, patterns and criteria that address characteristics such as modularity, encapsulation, loose coupling, separation of concerns, reuse and composability. Microsoft defines it as a loosely-coupled architecture designed to meet the business needs of the organization [35]. SOA is basically a collection of services. Every service is designed to fulfill a certain activity. Although, it is a loosely-coupled, two or more services could involve coordinating some activity dynamically to achieve a certain task using the concept of service composition. A service is a well-defined function that is self-contained and does not depend on the environment or state of other services. Service is a black-box for its consumers. A service may consist of underlying other services. So, service consumers need not worry about the inner logic of the service. Dynamically bound, loose coupling, self-adaptation, platform independent, highly modular, interoperability, discoverability are the common features of SOA.

For realization of SOA, the technology of Web Services is the simplest connection technology for implementing a loosely coupled SOA. SOA is an architectural style, whereas web service is a technology that can be used to implement on SOAs. Web service technology consists of several published standards SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and CORBA (Common Object Request Broker Architecture). As adopted in mobile technology, social media, cloud computing and big data analytics, SOA concept is gaining more popularity than ever before for providing integrated environments and systems from end to end. Oracle SOA Suite [36] is an example technology that simplifies connectivity by providing a unified experience to integrate across cloud, on-premise, and business to business.

Service provider creates a web service and deploys it to the service repository. A service provider can also be a service consumer. Service broker or service repository or service registry makes the information about available web service to the service consumer. Service requester or service consumer demands for a service or a set of services according to the need. They have to take it from the service brokers. Fig.1 illustrates a basic SOA structure.

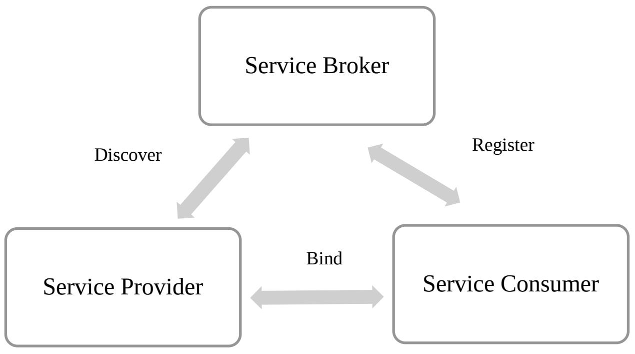

Fig.1. SOA structure

IV. Fault Handling Strategies of Services Oriented Computing

We have categorized fault handling strategies of service oriented computing into two categorized as local recovery strategies and global recovery strategies as shown in fig.2. Internal recovery concerns about the interactions among parameters in a service. Forward recovery technique is related with the transactional behaviors of the messages as results all or nothing. Backward recovery is associated with faults occurring situation where multiple services interact each other. It applies any one of the exception handling strategies like ignore, wait, retry, recompose, retryUntil etc. Backward recovery means rollback of the faulty service with the previous healthy version of the same service. Forward recovery is more optimized than backward recovery. This technique either ignore the fault service and goes forward to keep the rest of the system running with no harm or retry the faulty service again or substitute the faulty service with the equivalent other service which would be sufficient to fulfil the task of the current faulty service. Forward recovery has better performance than backward recovery and service recomposition and recreation.

V. Service-Oriented System Recovery Strategies

In this section, some of the service-oriented system recovery strategies are presented with their strengths and weaknesses. A strategy may be suitable for one condition of faulty service cannot be appropriate for another condition. So, fault repairing strategy depends on the nature of fault. Some of the popular fault handling strategies are discussed below.

a) *Ignore*: Ignore strategy just ignores the identified fault that does not affect the whole system and does not violates the goal. It is efficient action in case of performance utilization and reliable system if the fault is temporal.

b) *Replace*: During runtime any service can be faulty and cause a whole system failure with its QoS constraints violation. In case of service fault, replace action replaces the faulty service by the alternative equivalent service with the same functioning. The replacement action might call for compensation or rollback to recover. Replacement of faulty service by healthy service is time-consuming process but it enhances the reliability of the system.

c) *Retry*: Retry the fault generating service repeatedly till the maximum retry times. Web server is stateless between transactions; it does not maintain important state from first and last. The requests being processed are effectively dropped. Client may or may not receive complete relies to the in-process requests. The re-issuing of the request can lead to further problems since the same request may then be executed multiple times.

d) *Recompose*: this action searches for the alternative process with the same objective discarding the current faulty process. It may be the last option while repairing the faulty service because it is the most time-consuming fault handling strategy. But this strategy is reliable and suitable for all fault handling cases.

e) *Logging*: Logging mechanism is the popular mechanism to analyse the failure behaviour of various systems. Logging mechanism stores intercepted message traces of every transactions of service interactions. Later, if fault is occurred, message traces can be used for fault repairing purpose. It is challenging to recover from faulty condition using logging mechanism if service fault does not leave any trace in logs. Incompleteness and inaccuracy are two issues of logging-based fault repairing approach.

f) *Replication*: Replicating same service or process in several systems as backup requires additional resources and computational time. Fault service can be replaced by same version of service from backup system. N-version programming can be used to implement replication. In case of composite service, all candidate services must be available, if one of the services it consumes is unavailable, the main service may fail. Thus replication management is very challenging in this context. It is also challenging to reduce the number of replicas as required. This strategy has to detect updates in the environments automatically.

g) *Hybrid strategy* : A hybrid technique with application level logging and connection replication named CORAL (A Client-Transparent Fault-tolerant) [27] mechanism. CORAL recovers in-process requests and does not require deterministic servers or changes to the clients. To achieve the fault tolerance goals, active replication of the servers may be used, where every client request is processed by two (or more) server replicas. Logging of request is an alternative but two different replies for the same request may reach the client violating the requirement for transparent fault tolerance. Their approach has assumed only one host at a time can be affected by fault and the impact of the fault can be to either crash a process or crash or hang the entire host. Rollback and compensation are analogous to their usual definitions.

Some of the popular fault handling strategies are shown in fig. 2.

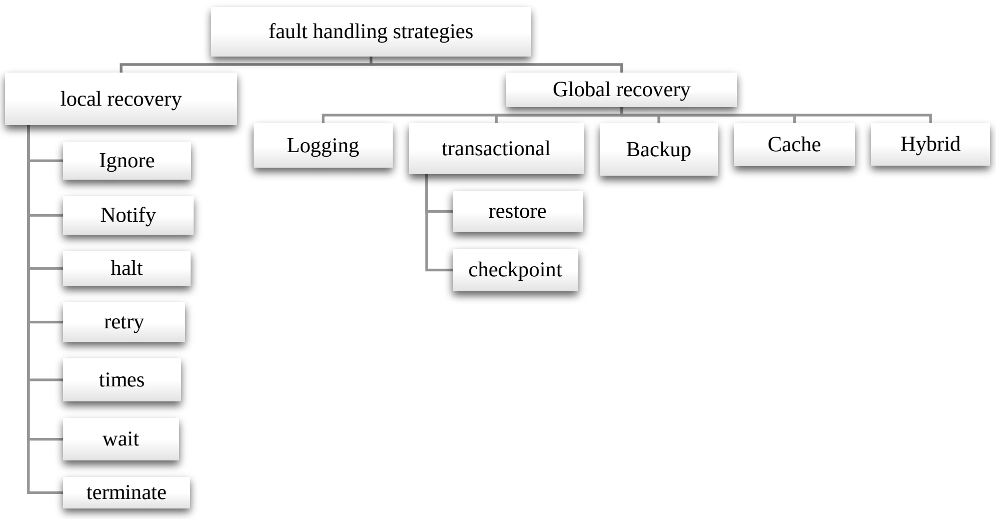

Fig.2. Fault handling strategies

A. *Challenges on applying the fault handling strategies:*

Several fault handling strategies with their major drawback and brief introduction are provided in table 1.

Table 1. Fault handling strategies and their drawbacks

Strategy	**Drawback**	**Details**
Replication	Time and space complexity (highly complex)	It is time-consuming to maintain same version of the service in several places and it takes storage space in several places.
Replace	Replacing faulty service requires overhead.	Optimal replacement policy[37] and service process reconfiguration [38] at code-level
Retry	Not suitable for pre-condition constraint violations	It is not suitable for those services to repair which are available for certain time period.
Recompose	– Most time-consuming approach, – It is the last option if any other strategies fail to repair faulty service.	Discards the faulty service and establishes an alternative process with the same goal.
Logging	– Extra message traffic – Service delay by message interception – Cost of agent deployment and execution – Cost of message storage and analysis	Re-issuing request leads same request may be executed multiple times.
Rollback	Different parts of the systems are owned by different institutions making it harder to perform true rollback [3].	-It is suitable for database technologies. -It would not be due to its unique features like interoperability and platform independent distributed system.

VI. Overview of Our Approach

As we have found performance as major challenge on handling fault in SOA, we have presented a model to address this challenge, and a priority model for Service-based System with three priority levels to handle the fault is proposed. All levels are corresponding with different fault handling strategies with their different purposes but we have adopted them here in holistic view irrespective of the service fault nature. We have proposed priority model with three levels of recovery strategies. First level has ignored, replace actions as fault handling strategies. Replace strategy here we used is optimal service replacement as proposed in [37]. Second level corresponds to wait and retry which has average performance rate. Level 3 resembles repeatUntil, recomposition, compensate actions, ranking components [39] as service oriented fault handling strategies. Level 1 is highly desirable and top priority level if fault occurs then FaultHandler will try to resolve the fault by ignore, and replace it by another equivalent substitute, which takes minimum time on handling service fault. Level 1 fault handling strategies have high performance rate. There is no any difference in fault reparability rate on these priority level. Our approach considerably has higher reparability rate because it deliberates on handling the fault via selecting the suitable strategy from higher level with best performance rate to lower level at least performance rate. Fig. 3 displays the fault handling strategy selection priority levels.

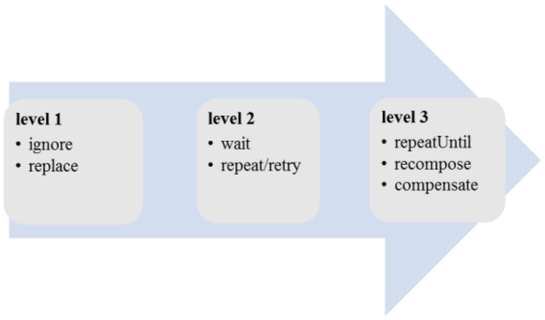

Fig.3. Level of fault handling strategies

A. *Level 1: fault handling strategies:*

Priority level 1 associates with ignore and replace actions as fault handling strategies. Replace strategy is optimal service replacement [37]. Level 1 is highly desirable and top priority level if fault occurs then Fault Handler will try to resolve the fault by ignore, or replace it by another equivalent substitute, which takes minimum time on handling the fault.

B. *Level 2: fault handling strategies:*

Second level corresponds to wait, retry or repeat, Microboot, logging and cache which have average performance rate. Firstly, system tries to recover the faulty service by the level 1 strategies if not possible only level 2 strategies could be the possible solutions for fault handling. In this level, fault handling process takes higher computational time but lower computation time than level 3 fault handling strategies.

C. *Level 3: fault handling strategies:*

Level 3 resembles repeatUntil, recomposition, compensation, reboot, hybrid and rollback actions as service oriented fault handling strategies. Level 3 fault handling strategies are the worst-case situation only when level1 and level 2 fault handling strategies are not possible to apply. Level 3 strategies are known as goal-preserving strategies. If the correct execution of the function is very crucial then this level strategies can be the best suitable condition to preserve the goal.

This approach can be applied at either platform-level or application level services in service-oriented computing. This approach starts from fault detection method and select the first priority level strategies promptly and if fault could not be handled by first level then fault handler selects the second level or intermediate priority level for average performance. As a final in the worst case, least level strategies are applied to resolve the faulty situation. The proposed approach fault handling technique can reduce the computational cost and increases the fault repairing rate.

- *Algorithm for fault repairing strategies selector:*

Algorithm 1, illustrates the working mechanism of strategy selector. The algorithm takes composite system *CS* and the system μ as input. We need to check the composite system to find the location of the faulty service and the impact region. Impact region should be expanded if the component services are associated with the faulty service as proposed by [7] and [40] . *Step 1.i)* detects faulty service S_f on composite service CS_k. *Step 1.j)* tries to apply first level repairing strategies. If fault repairing process is success by ignoring the faulty service then the further execution continues. Otherwise, it applies replace strategy where faulty service S_f is replaced with equivalent substitute service S_h which is a healthy service. *Step 1.k)* concerns with level 2 strategies where firstly retry the same service again to check whether it is able to fulfil the same need or not if fulfilled successfully then workflow executes the next statement other wise it waits T_x milliseconds time as in *Step 1.k)b*. If level 2 strategies are also failed to accomplish the fault repairing task then it applies *Step 1.l)* level 3 stragies to repair the fault condition. In level 3, Firstly it tries with RepeatUntil action for certain time say $y{*}T_x$ (y times T_x). If repeating the same service again and again for fix time duration achieved the goal then repairing process ends

Algorithm 1 for fault repairing strategies selector

```
INPUT: CS, μ
1) For each candidate of composite service $CS_k \in \mu$, $1 \le k \ge m$;
    a) For all i, $1 \le i \ge n$;
      i) Detect faulty service ($S_f$, $CS_k$);
      j) Apply level1 strategies
         a. Ignore $S_f$, $1 \le f \ge n$, if fault repairing is success then go
            to Step 3.
         b. Replace $S_f$ of $C_i$ by equivalent $S_h$ where $1 \le h \ge n$. if fault
            repairing is success then go to Step 2.
      k) Apply level2 strategies
         a. Retry $S_f$ again $1 \le f \ge n$, if fault repairing is success then
            go to Step 2.
         b. Wait for time $T_x$ milliseconds;
      l) Apply level3 strategies
         a. RepeatUntil, certain time $y{*}T_x$ (y times $T_x$), if repairing
            is desired after y times of $T_x$ then go to Step 2.
         b. Call any compensation strategies like transactional
            atomicity, logging, replication etc. for repairing purpose
            and after repairing successfully go to Step 2.
         c. Recompose $S_f$, where $1 \le f \ge n$ and $S_f$ is $CS_i \in \mu$
2) Continue the execution
3) Stop
```

successfully and continues the execution with the next statement. Otherwise it calls compensation strategies like trasactional atomicity, logging, replication etc. for repairing purpose. As a final but reliable strategy as mentioned in *Step 1.l)c*, a service is recomposed at a execution time and tries to undertake the duty of faulty service to accomplish the faulty task.

VII. Performance Analysis

In this section, performance overhead has been calculated with different workloads. We collect performance on the basis of computation time, average response time and fault detection rate without our approach and with our approach. The result of the experiment is shown in fig. 4 that shows performance analysis with computation time in milliseconds over different priority levels of fault handling strategies with respect to the number of composite services. Overall, the computational time of the level 1 strategies is considerably less among all levels for any number of composite services. However, till 6 composite service, computational time of level 1 and level 2 strategies seems idle. Afterward level 2 takes slightly more computational time. Level 3 strategies appear worst case strategies in terms of computational time among all strategies.

A. The comparison of computation time between with and without the approach

The performance analysis with the comparison in terms of computation time in milliseconds over our proposed approach and without our approach with respect to the number of composite services is depicted in Fig.5. Generally, the computational time of our approach is considerably less than without our approach irrespective of the number of composite services. However, there is less variation when there is fault in less number of composite services, as number of composite services increased it becomes complex to handle the fault and takes a lot of computation time.

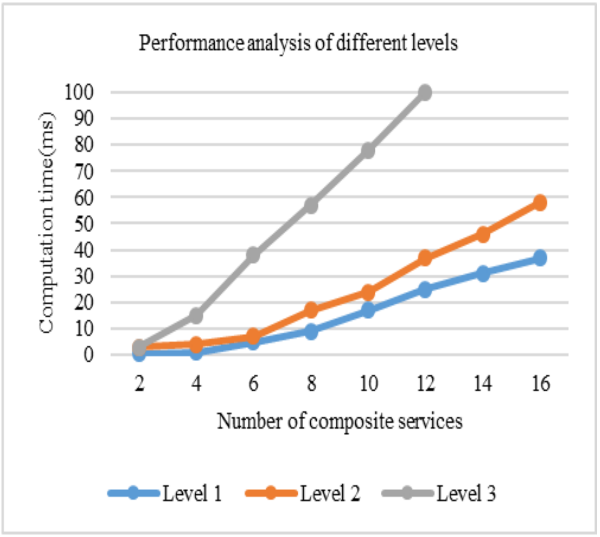

Fig.4. Performance analysis of different priority levels

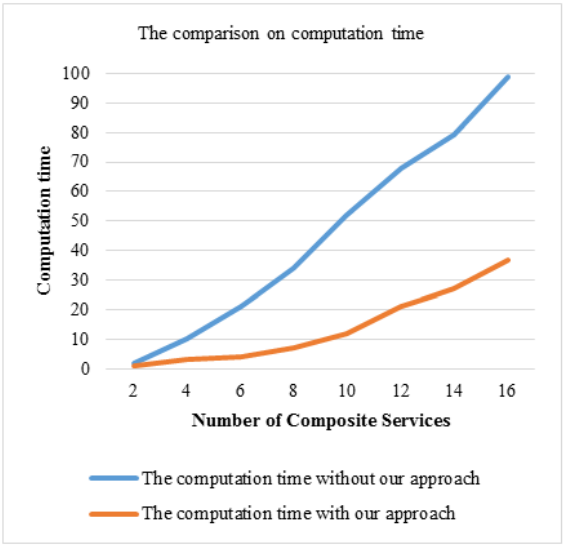

Fig.5. The comparison of computation time with and without our approach

B. The comparison on average response time between with our approach and without the approach

Comparison of average response time in milliseconds of our approach and without our approach over various number of composite services is shown in Fig.6. Overall, the average response time of our approach is considerably less service oriented computing system without our approach regardless of number of composite services.

C. The comparison of fault repairing rate between with our approach and without the approach

Reliability on fault handling of Service Oriented Computing system is very big challenge to achieve. Performance ensures the reliability. Fault repairing rate is the probability of repairing the faulty service. Fig. 7, shows comparison of fault repairing rate between our approach and without our approach with respect to the number of test suites. Our approach for fault handing of service computing system has significant fault repairing rate.

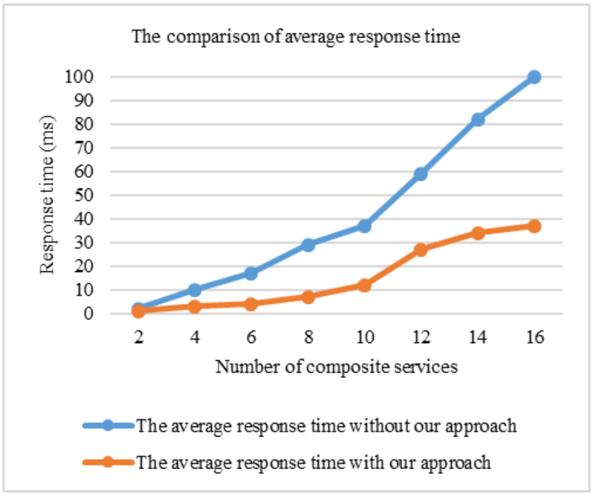

Fig.6. Comparison of average response time between with our approach and without our approach

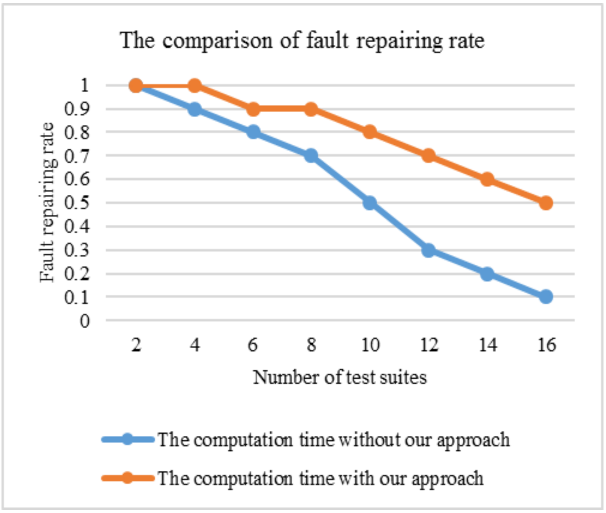

Fig.7. The comparison of fault repairing rate between with our approach and without our approach

VIII. Conclusion

In this paper, a priority based fault handling strategy selecting approach for fault recovery is proposed using priority selector and fault handler. This approach can be applied at either platform-level or application level services in service-oriented computing. This approach starts from fault detection method and select the first priority level strategies promptly and if fault could not be handled by first level then fault handler selects the second level or intermediate level priority level for average performance. As a final in the worst case, least level strategies are applied to resolve the faulty situation. The proposed approach fault handling technique is optimized and reduces the computational cost and complexity. Through several experiments, the correctness of our algorithms and the efficiency of our approach is proved. This approach results better performance and high rate of fault repairing than without our approach.

Acknowledgments

The authors would like to thank ICCR, Ministry of Foreign Affairs, India (Silver Jubilee Scholarship Scheme) for providing funds and DST-CIMS, Institute of Science, BHU, India for providing necessary infrastructure and facilities for undertaking this research work.

References

[1] H. J. La and S. D. Kim, "Static and dynamic adaptations for service-based systems," *Inf. Softw. Technol.*, vol. 53, no. 12, pp. 1275–1296, 2010.

[2] M. Wang, K. Y. Bandara, and C. Pahl, "Integrated constraint violation handling for dynamic service composition," *SCC 2009 - 2009 IEEE Int. Conf. Serv. Comput.*, pp. 168–175, 2009.

[3] L. Baresi and S. Guinea, "Self-Supervising BPEL Processes," *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 247–263, 2011.

[4] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, "Exception handling for repair in service-based processes," *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 198–215, 2010.

[5] D. Ruan and S. Lu, "Task exception handling in the VIEW scientific workflow system," *Proc. - 2010 IEEE 7th Int. Conf. Serv. Comput. SCC 2010*, pp. 637–638, 2010.

[6] S. Brüning, S. Weißleder, and M. Malek, "A fault taxonomy for service-oriented architecture," *Proc. IEEE Int. Symp. High Assur. Syst. Eng.*, pp. 367–368, 2007.

[7] A. Ismail, J. Yan, and J. Shen, "Analyzing fault-impact region of composite service for supporting fault handling process," *Proc. - 2011 IEEE Int. Conf. Serv. Comput. SCC 2011*, pp. 290–297, 2011.

[8] L. Wang, A. Wombacher, L. F. Pires, M. J. Sinderen, and C. Chi, "Robust client/server shared state interactions of collaborative process with system crash and network failures," *Proc. - IEEE 10th Int. Conf. Serv. Comput. SCC 2013*, pp. 192–199, 2013.

[9] H. Chai and W. Zhao, "Byzantine Fault Tolerance for Services with Commutative Operations," *2014 IEEE Int. Conf. Serv. Comput.*, pp. 219–226, 2014.

[10] X. Zhu, C. He, R. Ge, and P. Lu, "Boosting adaptivity of fault-tolerant scheduling for real-time tasks with service requirements on clusters," *J. Syst. Softw.*, vol. 84, no. 10, pp. 1708–1716, 2011.

[11] M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino, "Semantics-Based Design for Secure Web Services," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 33–49, 2008.

[12] C. M. Tang, W. K. Chan, and Y. T. Yu, "Extending the Theoretical Fault Localizaton Effectiveness Hierarchy with Empirical Results at Different Code Abstraction Levels *," in *IEEE 38th Annual International Computers, Software and Applications Conference Extending*, 2014, pp. 161–170.

[13] B. R. Kandukuri, R. P. V., and A. Rakshit, "Cloud Security Issues," *Proc. 2009 IEEE Int. Conf. Serv. Comput.*, pp. 517–520, 2009.

[14] F. Balbastro, A. Capozucca, and N. Guelfi, "Analysis and framework-based design of a fault-tolerant web information system for m-health," *Serv. Oriented Comput. Appl.*, vol. 2, no. 2–3, pp. 111–144, 2008.

[15] M. Sama, D. S. Rosenblum, Z. Wang, and S. Elbaum, "Multi-layer faults in the architectures of mobile, context-aware adaptive applications," *J. Syst. Softw.*, vol. 83, no. 6, pp. 906–914, 2010.

[16] J. Chen, Q. Li, C. Mao, D. Towey, Y. Zhan, and H. Wang, "A Web services vulnerability testing approach based on combinatorial mutation and SOAP message mutation," *Serv. Oriented Comput. Appl.*, vol. 8, no. 1, pp. 1–13, 2014.

[17] V. Garousi, L. C. Briand, and Y. Labiche, "Traffic-aware stress testing of distributed real-time systems based on UML models using genetic algorithms," *J. Syst. Softw.*, vol. 81, no. 2, pp. 161–185, 2008.

[18] S. Basu, F. Casati, and F. Daniel, "Toward web service dependency discovery for SOA management," *Proc. - 2008 IEEE Int. Conf. Serv. Comput. SCC 2008*, vol. 2, pp. 422–429, 2008.

[19] J. Zhang, Y. C. Chang, and K. J. Lin, "A dependency matrix based framework for QoS diagnosis in SOA," *IEEE Int. Conf. Serv. Comput. Appl. SOCA' 09*, vol. 0, no. c, pp. 299–306, 2009.

[20] J. Zhang, X. Zhang, and K. J. Lin, "An efficient Bayesian diagnosis for QoS management in service-oriented architecture," *Proc. - 2011 IEEE Int. Conf. Serv. Comput. Appl. SOCA 2011*, 2011.

[21] Q. He, J. Han, Y. Yang, J. G. Schneider, H. Jin, and S. Versteeg, “Probabilistic critical path identification for cost-effective monitoring of service-based systems,” *Proc. - 2012 IEEE 9th Int. Conf. Serv. Comput. SCC 2012*, pp. 178–185, 2012.

[22] C. A. Sun, Y. M. Zhai, Y. Shang, and Z. Zhang, “BPELDebugger: An effective BPEL-specific fault localization framework,” *Inf. Softw. Technol.*, vol. 55, no. 12, pp. 2140–2153, 2013.

[23] Y. Dai, L. Yang, B. Zhang, and Z. Zhu, “Exception diagnosis for composite service based on error propagation degree,” *Proc. - 2011 IEEE Int. Conf. Serv. Comput. SCC 2011*, pp. 160–167, 2011.

[24] C. Ye and H. A. Jacobsen, “Whitening SOA testing via event exposure,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1444–1465, 2013.

[25] L. Waszniowski, J. Krákora, and Z. Hanzálek, “Case study on distributed and fault tolerant system modeling based on timed automata,” *J. Syst. Softw.*, vol. 82, no. 10, pp. 1678–1694, 2009.

[26] A. Benharref, R. Dssouli, M. A. Serhani, and R. Glitho, “Efficient traces’ collection mechanisms for passive testing of Web Services,” *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 362–374, 2009.

[27] N. Aghdaie and Y. Tamir, “CoRAL: A transparent fault-tolerant web service,” *J. Syst. Softw.*, vol. 82, no. 1, pp. 131–143, 2009.

[28] M. Cinque, D. Cotroneo, and A. Pecchia, “Event logs for the analysis of software failures: A rule-based approach,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 806–821, 2013.

[29] D. Jayasinghe, C. Pu, F. Oliveira, F. Rosenberg, and T. Eilam, “AESON: A model-driven and fault tolerant composite deployment runtime for IaaS clouds,” *Proc. - IEEE 10th Int. Conf. Serv. Comput. SCC 2013*, pp. 575–582, 2013.

[30] J. Cubo, M. Sama, F. Raimondi, and D. Rosenblum, “A model to design and verify context-aware adaptive service composition,” *SCC 2009 - 2009 IEEE Int. Conf. Serv. Comput.*, pp. 184–191, 2009.

[31] F. Belli and M. Linschulte, “Event-driven modeling and testing of real-time web services,” *Serv. Oriented Comput. Appl.*, vol. 4, no. 1, pp. 3–15, 2010.

[32] V. Ermagan, I. Krüger, and M. Menarini, “A fault tolerance approach for enterprise applications,” *Proc. - 2008 IEEE Int. Conf. Serv. Comput. SCC 2008*, vol. 2, pp. 63–72, 2008.

[33] M. Jensen, “A fault propagation approach for highly distributed service compositions,” *Proc. - 2008 IEEE Int. Conf. Serv. Comput. SCC 2008*, vol. 2, pp. 507–510, 2008.

[34] IBM, “Service Oriented Architecture(SOA): Simply good design,” *IBM*, 2016. [Online]. Available: https://www-01.ibm.com/software/solutions/soa/. [Accessed: 25-Dec-2016].

[35] D. Linthicum, “Chapter 1: Service Oriented Architecture (SOA),” *Microsoft*, 2016. [Online]. Available: https://msdn.microsoft.com/en-in/library/bb833022.aspx. [Accessed: 23-Dec-2016].

[36] Oracle, “Oracle SOA : Service Oriented Architecture Engineered to Adapt,” *Oracle*, 2016. [Online]. Available: http://www.oracle.com/us/products/middleware/soa/overview/index.html. [Accessed: 10-Dec-2016].

[37] S. S. Pillai and N. C. Narendra, “Optimal Replacement Policy of Services Based on Markov Decision Process,” *2009 Ieee Int. Conf. Serv. Comput.*, pp. 176–183, 2009.

[38] K. J. Lin, J. Zhang, Y. Zhai, and B. Xu, “The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA,” *Serv. Oriented Comput. Appl.*, vol. 4, no. 3, pp. 157–168, 2010.

[39] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, “Component Ranking for Fault-Tolerant Cloud Applications,” *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 540–550, 2012.

[40] A. Ismail, J. Yan, and J. Shen, “Incremental service level agreements violation handling with time impact analysis,” *J. Syst. Softw.*, vol. 86, no. 6, pp. 1530–1544, 2013.

Authors’ Profiles

Guru Prasad Bhandari is working as a Research Scholar at DST-CIMS, Institute of Science, Banaras Hindu University, Varanasi, India. His research area is service-oriented computing and reliability analysis. He is currently working on ‘Fault Analysis of Service-oriented Computing’. He is pursuing his doctoral work under the supervision of Dr. Ratneshwer.

Ratneshwer did his Ph.D. in Component Based Software Engineering from Indian Institute of Technology, Banaras Hindu University, Varanasi (IIT-BHU), India. His research area is CBSE and SOA. He is serving as an Assistant Professor in Department of Computer Science (MMV), Banaras Hindu University, India. He is actively involved in teaching and research for last 8 years. One research monograph is published by LAP Germany and one book chapter has been published by IGI Global Publication. He has 16 research papers in International journals and 16 research papers in international/national conference proceedings in his credit.

How to cite this paper: Guru Prasad Bhandari, Ratneshwer,"Fault Repairing Strategy Selector for Service-Oriented Architecture", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.6, pp.32-39, 2017.DOI: 10.5815/ijmecs.2017.06.05