# Proactive Exploratory Testing Methodology During Enterprise Application Modernization

Ravikumar Ramadoss[#1], N.M.Elango[#2]

[#1] Research Scholar, Sathyabama University, Jeppiar Nagar, 600119, India,
ravikumar_rrk@hotmail.com

[#2] Dept. Of Computer Applications, R.M.K Engineering College, Kavaraipettai, 601206, India,
hod.mca@rmkec.ac.in

**Abstract— Traditional web applications are getting modernized to meet the expectations of customer demands. Various features like channels, infrastructure enhancements, cloud and big data adoptions, business analytics are getting implemented. This leads to complexity of the target application and also leads to difficulty in meeting the desired SLA. A tester can proactively identify issues by performing exploratory testing. This will act as proactive approach to simulate the performance testing during coding stage itself. In this paper, we use such an exploratory testing approach to test the enterprise modernized web application. We find out that the proposed approach is effective in bringing out major potential bottlenecks and predict how the SLA will get impacted during the simulation process.**

**Keyword-**Enterprise Modernization, Dynamic Spike, Exploratory Testing, Proactive Performance Engineering

## I. INTRODUCTION

Enterprise modernized application has a stringent SLA definitions. As the complexities in terms of the characteristics are extended to meet the expectations of the customer, meeting the desired SLA is must for enterprise. In this paper we propose a framework for exploratory testing by following a dynamic spike injection pattern. This approach has various advantages:

Traditionally application will get functional verification and if any performance issues are getting reported, then those transactions will undergo reactive engineering to tune it well or proactive performance engineering gets followed to verify the transaction performance, but even the proactive approach also done after the coding and functional verification. On the other hand, the proposed approach validates the performance of the application even during the development mode to pin-point any major performance issues. In this paper

We are discussing a testing strategy where proactive benchmark analysis on the functionally verified code to report any transaction analysis.

We discuss the proposed exploratory testing approach to proactively validate the performance of the transaction during the coding life cycle itself.

We also perform dynamic spike injection approach to identify the issues which impacts the performance SLA. To conclude, we discuss the simulation experiments we have conducted to test the sample enterprise modernized web application to proactively discover issues. The rest of the paper is organized as follows:

Concepts involved in exploratory testing strategy are introduced in section II. Section III discuss the related work happened in this area. In section IV we discuss about the proposed exploratory testing approach, its architecture models along with challenges and technology stack used to implement the modernized application. In section V we discuss about the experiments and how we identified the issues happening inside the transactions which are not meeting the desired SLA. Section VI discuss about the performance testing we conducted to validate the approach. Finally we discuss about the conclusion and future work.

## II. CONCEPTS INOVLVED IN PROACTIVE EXPLORATION TESTING STRATEGY

In order to successfully modernize enterprise applications, it is necessary to test and validate the performance aspect of the target application. Before discussing about the proposed approach lets discuss first the important aspects which are used in this paper.

### *A. Exploratory Testing*

Tester applies innovative approach by applying his creativity to generate test cases. It is part of the black box testing approach. These testing approaches can be applied in any stage of the development process. The key is to explore non captured test cases to enhance the quality of the solution getting implemented.

*B. Dynamic Spike Injection*

Code can be non-intrusively appended into the existing functionally stabilized code to test the dynamic behaviour of the application. Spikes can be created which are virtual in nature, which also has an impact on major necessary attributes contributes to the performance SLA.In traditional approach if there is an issue the performance tuning of the application can be done via reactive approach, or proactively conduct performance test and tune the code which impacts the transaction functionality [9]. In proposed exploratory approach testers can simulate the performance behaviour of the application by doing the following dynamic spike injections which will cause major impact on the session, memory, and heap and thread locks and response time delay. The key thing to observe here is the performance tuning will happen parallel along with the actual code generation.

*C. Session Spike*

Enterprise web application keep navigation and user even information in multiple data structures and get pushed to the HTTP session. Session spike intrudes into the existing code and increase the session size as per the session scope defined in the code flow. This will substantially increase the memory foot print per page.

*D. Memory Spike*

To introduce force memory leaks into the code which impacts the SLA of the transaction and also to impact the garbage collection process of the well performing JVM.

*E. Heap and Thread Locks*

Increase the heap memory by introducing the force cache to disturb the heap settings. Programmatically hold the thread lock and cause thread dead locks to forcefully impact the execution of the transaction.

*F. Response Time Delay*

Based on the above spike injections and also the thread contention issues which makes the execution of the transaction to sleep state and later get recovered based on the defined timelines to cause impact on the SLA of the performing transaction.

*G. Aspect Oriented*

Non-Intrusive code changes get applied into the existing code base via aspect oriented programming (AOP). Using AOP we will dynamically define the point cut expressions (aop: pointcut) for the respective aop: aspect. The aop aspects get injected into the code base on aop: before and aop: after entries.

## III.RELATED WORK

During enterprise web application modernization, proactive performance engineering along with functional testing is the generic practices for studying the performance SLA of the target application. Traditional testing methodologies leverage white box code analysis tools such as CAST [17] which scan through the entire code base and provide detailed report on various metrics. The performance testing tools like IBM Rational Performance Tester or Web Performance Suite or Apache JMeter [18] perform different load tests and also help in getting the benchmark analysis report.

Software fault injection uses different techniques to inject faults into the system. For instance Xception [7] uses registers on the hardware and enable debugging. Ferrari [5] uses dynamic injection capabilities and corrupt messages to inject faults were also done [6]. Both these techniques can be combined together. The other injection techniques such as NFTAPE [4] allows inversion causes, delays, and other spike related techniques to demonstrate the injection capabilities. Orchestra [11] has the probing technique along with software implemented fault injector (SWIFI) has more complex models for session, memory and process crashes as explained in the image application.

We want to achieve the exploratory testing as non-intrusive without impacting the enterprise modernization characteristics along with validating the SLA of the modernized application [8]. We also avoided having any dependency with third party libraries and hence made it license friendly to test it on all different applications. Aspect oriented programming as a concept is generic in nature and the same can be applied to any format of application. It can be plugged into any existing application without changing the code base.

## IV.PROPOSED EXPLORATORY TESTING APPROACH

In order to efficiently validate the approach on the enterprise modernized application, it requires unique and efficient strategy in phased manner. The application taken for the proposed approach plays a very crucial role in performing the exploratory testing. The following section will discuss about the modernized application, and the phase wise approach.

*A. Architecture Models in Enterprise Modernized Application*

The enterprise web application is based on J2EE tech stacks, which has the logical layers as action, application, data access, web service consumption respectively [1], [10]. In our approach we have chosen three models as defined in Fig. 1. This model has the flexibility to test the strategy on with in network, end point calls which are outside of the network and also high availability testing during the process crashes.

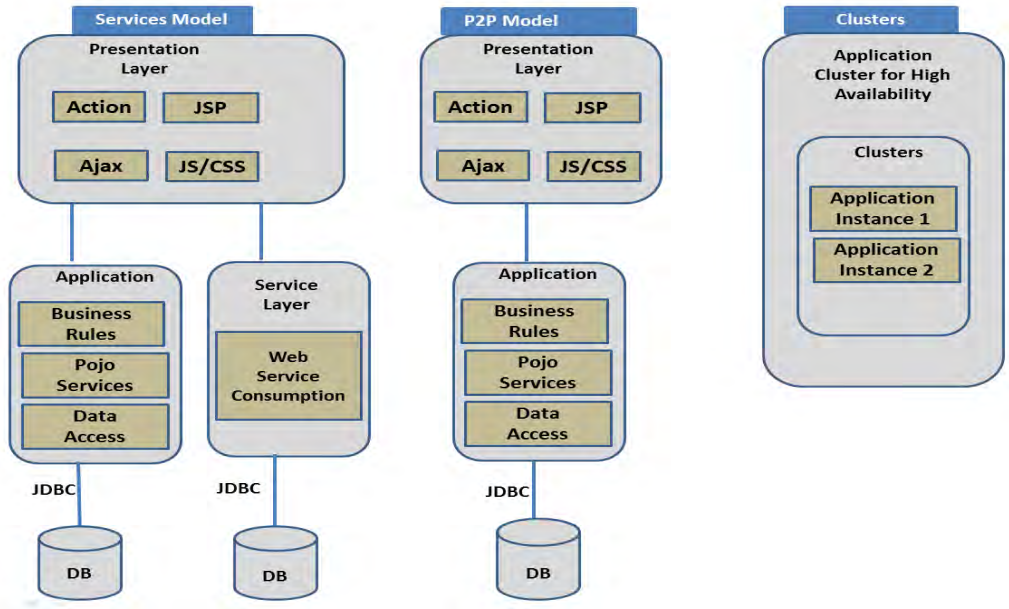


Fig. 1. Enterprise Modernized Application Architecture Models

SOA mode: Here the application is implemented in logical layers but with external and internal interfaces integration via rest/SOAP based API calls. This model is needed to simulate the external interfaces integration.

P2P Mode: Here the application is implemented in logical layers and the interactions are within the network and also the database interactions are happening within the same network. It is purposely made as one instance based application.

High availability mode: Here the application is deployed in cluster environment to simulate the session, memory heap and thread related spikes. This environment is also needed to simulate different workload configurations.

*B. Challenges and Major Enterprise Modernization Characteristics*

Here we are discussing about the major modernization characteristics of the enterprise modernized web application. These aspects are crucial in making the application to meet the expected customer demands.

*1) Channels:* Traditionally modernized application gets accessed from web browsers, and after the modernized characteristics it needs to be accessed from multiple sources like channels, ivr, sms, text, mobile and other smart devices [14].

2) *Context aware filtering:* As the characteristics are getting extended, the application generates huge volume and variety of data. Hence only the relevance data have to be filtered out to get persisted in to the NOSQL database.

3) *Preservation of existing business knowledge:* Preservation of existing business knowledge: SME team who aware of the traditional application business as well as technical knowledge is limited and hence preserving the existing functional knowledge via various log analysis, SME interviews have to happen.

4) *Hybrid cloud adoption:* Traditional applications are getting deployed as in house data center and on premise and the scalability are mostly by doing horizontal or vertical capacity related scaling. On-demand scalability needs to be added to the target application by keeping the privacy and security concerns intact. The hybrid cloud adoption provides those capabilities to application [3], [6], [12].

5) *Insight modelling:* Traditional application does only ad-hoc business reporting, this needs to be extended to have dynamic unstructured business insight modelling to yield desired results from the volume and variety of data getting generated [2].

6) *Technology adoptions:* Traditional application has Service oriented architecture (SOA) but it's not enough to withstand the external interfaces and other integration needs. It has to be modernized with SOA suites to enhance the capability [13].

*C. Phase Wise Approach*

In order to efficiently validate our approach the testing process has to be defined in multiple phases. The major purpose of dividing the approach into multiple phases is to:

- Identify the complexity of various transactions involved with in the application.
- Filter out which transaction is more critical to analysis.
- Identify the enterprise modernization characteristics to simulate the dynamic spike injections.
- Identify different ways to cause error or halt in the code execution flows.

The various phases in exploratory testing are:

Phase 1: Collection of existing transaction characteristics along with SME defined SLA numbers.

Phase 2: Perform functional testing.

Phase 3: Perform benchmark analysis by simulating the stabilized functional tested application whether it is performing it needs.

Phase 4: If the benchmark analysis SLA numbers for any of the transaction is not meeting the desired SLA, perform proactive performance engineering [15] based code analysis to fix the issue. Then proceed to phase 5.

Phase 5: In order to validate the system during spike load in presence of various faults [11], the following is the approach:

- Perform the same benchmark analysis by applying same workload patterns.
- During transaction execution, inject various faults which can cause dynamic spike into the system.
- Monitor various metrics like memory, session, JVM process, heap and threads and also analyze system logs for any major severity issues getting reported.
- Perform functional testing to check the system behaviour.
- Identify the issues where functionality is broken and also perform performance engineering steps.

## V. EXPLORATORY TESTING EXPERIMENTS

In this section, we are exploring various exploratory test cases and scenarios. For each scenarios we will present the existing scenario, exploratory scenario, non-intrusive changes to the codebase, outcome observation, the impact on the performance and also the code changes for before and after exploratory changes.

*A. Test Scenario 1 : Session Spike*

The strategy is to increase the session usage for the particular scenario and observe the impact the session creates. The session might have the impact on the other navigations as the scope of the session varies between the flows. The observation has to be extended to all the previous and next flows.

The various test exploratory testing scenarios are explained in Table I.

TABLE I
Session Spike Scenario

| **Session Spike** | **Description** |
|---|---|
| Existing Scenario | Moderate amount of Session storage |
| Exploratory Scenario | Increase the session storage with change in the session identifier per scope of the page |
| Non-intrusive change | The session identifier api is non-intrusively plugged in to store the same session object with different dynamically generated identifiers in incremental pattern |
| Observation | The subsequent navigational flows get disturbed as the session parameters are getting retrieved in that flow |
| Performance impact | The page response time is increased from the benchmark data. The resources getting loaded on to the page was also got delayed |

The dynamic expressions gets executed at runtime and the aop listeners gets triggered to apply the required logic based on the test configurations. The implementation gets executed only when the classpath expressions are getting matched with the aop: pointcut expressions. The original state of the code and the aop expressions of session spike simulation and the actual implementation of the same are shown in Table II.

TABLE II
Non-Intrusive Code for Session Spike Simulation

| **Existing Code** | **Non-Intrusive aop definition** | **Generic Session Spike Code** |
|---|---|---|
| Public void getInvoiceData()<br>{<br>If(null!=session ){<br><br>InvoiceSearchVO searchVO = buildInvoiceSearchVO();<br>InvoiceDataVO invoiceDataVO = service.getInvoiceData(invoiceSearchVO);<br><br>session.put("invoiceData", invoiceDataVO);<br>}<br>} | <aop: pointcut id="servicePointcut" expression="execution (* com.webmod.sessionsimulation..*.*(..))"/><br><br><br><aop:aspect id="sessionServiceAspect" ref="sessionSimulation"><br><aop:before method="sessionEntry" pointcut-ref="servicePointcut"/><br><aop:after-returning method="sessionExit" returning="result" pointcut-ref="servicePointcut"/><br><aop:after-throwing pointcut-ref="servicePointcut" throwing="exception" method="sessionAfterThrowing"/><br></aop:aspect><br><br><bean id="sessionSpike" class="com.webmod.sessionsimulation.SessionSpike"/> | public void sessionEntry(JoinPoint joinPoint) {<br>Object[] args = joinPoint.getArgs();<br>String name = joinPoint.getSignature().toLongString();<br>StringBuffer sb = new StringBuffer(name + " called with: [");<br>log.debug("Inside Method entry");<br>InoviceData invoiceData = args[0];<br>for(int simulationSize=0;simulationSize<=100;simulationSize++){<br>String uniqueSessionId = getUniqueId();<br><br>session.put(uniqueSessionId,inoviceData);<br><br>}<br><br>} |

*B. Test Scenario 2 : Heap and Thread Locks*

The strategy is to increase the heap size usage when the transaction is in execution and parallel apply forceful thread lock to cause un-expected delay into the system. The observed expectation needs well placed monitoring strategy cut-across application container, its internal JVM processes [15], [16]. The various test exploratory testing scenarios are explained in Table III.

TABLE III
Heap and Thread Locks Scenario

| **Heap and Thread Lock** | **Description** |
|---|---|
| Existing Scenario | Moderate amount of heap usage and no thread lock reported |
| Exploratory Scenario | To increase in amount of heap usage and introduce thread locks dynamically |
| Non-intrusive change | Memory leak and thread lock api are dynamically introduced during the execution of the transaction. |
| Observation | Sudden spike in the memory foot print and hung threads getting reported and also severity warning in the system |
| Performance impact | Concurrent requests to the transaction have to wait for the current hanging thread to finish and also the heap memory extends the garbage cycle process. |

The original state of the code before heap and thread locks and the aop expressions for lock and thread simulations and the actual implementation of the aop expression are shown in Table IV.

TABLE IV
Non-Intrusive Code for Heap and Thread Locks Simulation

| **Existing Code** | **Non-Intrusive AOP definition** | **Generic Heap and Thread lock AOP Code** |
|---|---|---|
| Public void rateCalculation()<br>{<br><br>RateVO rateVO = service.performRateCalculation();<br><br>applyRateRules(rateVO);<br>} | <aop: pointcut id="servicePointcut" expression="execution (* com.webmod.heapThreadsimulation..*.*(..))"/><br><br><aop:aspect id="sessionServiceAspect" ref="heapThreadSimulation"><br><aop:before method="heapThreadEntry" pointcut-ref="servicePointcut"/><br><aop:after-returning method="heapThreadExit" returning="result" pointcut-ref="servicePointcut"/><br><aop:after-throwing pointcut-ref="servicePointcut" throwing="exception" method="heapThreadAfterThrowing"/><br></aop:aspect><br><br><bean id="heapThreadLock" class="com.webmod.sessionsimulation.HeapThreadLock"/> | public void sessionEntry(JoinPoint joinPoint) {<br>Object[] args = joinPoint.getArgs();<br>String name = joinPoint.getSignature().toLongString();<br>StringBuffer sb = new StringBuffer(name + " called with: [");<br>log.debug("Inside Method entry");<br>InoviceData invoiceData = args[0];<br><br>Thread t = getCurrentThread();<br>t.lock();<br>performHeapOperation();<br>if(threadLockReleased(t))<br>{<br>RateVO rateVO = service.performRateCalculation();<br>applyRateRules(rateVO);<br>}<br>} |

*C. Test Scenario 3 : Enterprise Application Characteristics Crash Testing*

The strategy is to dynamically crash the database instances by leveraging the process signals. The observation is done at the application container level and also the data integrity check conducted at the database level. The observation predicts the data integrity issue to be taken care by the application team in their implementation scenarios. The various test exploratory testing scenarios are explained in Table V.

TABLE V
Enterprise Application Characteristics Crash Simulation

| **Enterprise Application Characteristics Crash** | **Description** |
|---|---|
| Existing Scenario | All database instances are up and running properly. |
| Exploratory Scenario | Dynamically crash and kill the database instance to record the impact |
| Non-intrusive change | Dynamically crash and kill the database from the operating system process. |
| Observation | The application stability was not implemented properly and also the data integrity related issues are also recorded. |
| Performance impact | The application was not able to respond immediately as there is a down in most of the enterprise modernization characteristics |

Identification of the database instance process id and send the dynamic crash signal by issuing the following command. Identify the appropriate process id from the environment where the instance gets deployed and issue the following command to crash the instance without sending any notification signals to the applications where it is connected to. Triggers are also placed in the database environment to capture the data integrity related data swaps happened at the table space.

kill -9 $(ps aux | grep EnterpriseWebMod | grep -v grep | awk '{print} ')

## VI. PERFORMANCE VALIDATION OF THE PROPOSED APPROACH

The goal is to validate the proposed exploratory testing approach. In order to validate the same, we have considered the same transaction characteristics which we have identified for tester based model for the performance simulation. As the proposed approach is non-intrusive in nature, we were able to apply the exploratory methodology without impacting the execution cycle of the transaction. The major objective is to validate how the SLA numbers are turning about based on page rendering time between both the approaches. We have carefully considered the workload model of the transaction so that the results obtained are not having network latency or choke which can impact the final SLA numbers.

### *A. Performance Testing Strategy Definition*

The strategy is to validate the SLA by performing a load testing on the modernized application. For this purpose, we deployed the application on clustered environment. The clustered environment provides flexibility to withstand the spike injected on to the application. The session spikes, heap and thread locks and enterprise modernization crash were considered even in the performance test of the application. The toggle features via AOP has the flexibility in applying the dynamic spikes into the application on every simulation of performance tests. The code base in which the exploratory testing code was dynamically injected along with transaction characteristics are shown in Table VI.

### *B. Exploratory SLA vs Performance Testing SLA*

The exploratory testing SLA was captured using the browser rendering time end to end. The average transaction time of the identified transactions with the defined workloads are shown in Fig. 2. The graphical representation by comparing the SLA numbers derived from exploratory SLA and performance SLA as shown in Fig. 3. From the experimental results it shows that the SLA numbers derived from performance testing are almost 95 percentile matching to the proposed approach as shown in Table VII.

TABLE VI
Transaction and its Characteristics

| | **Characteristics** | | | | **Non-Intrusive code applied during performance test** | | |
|---|---|---|---|---|---|---|---|
| **Transaction Name** | **Database Calls** | **Business Rules** | **External Interface calls** | **Modernization component calls** | **Session Spikes** | **Heap and Thread Locks** | **Enterprise Modernization crash** |
| Role-based Authorization (T1) | 3 | 2 | 0 | 2 | Y | N | N |
| Reporting and Analytics (T2) | 3 | 1 | 3 | 1 | N | Y | N |
| Rate calculation via channels (T3) | 6 | 9 | 3 | 3 | Y | Y | Y |
| Payment interface integration (T4) | 5 | 3 | 0 | 5 | N | Y | Y |
| Invoice processing (T5) | 9 | 5 | 4 | 7 | Y | Y | Y |

TABLE VII
SLA validation with the Proposed Approach

| Transaction Name | Complexity | SME Defined SLA | Exploratory Testing Derived SLA | Performance Test Derived SLA |
|---|---|---|---|---|
| Role based Authorization (T1) | Medium | 2.5 | 3.5 | 3.8 |
| Reporting and Analytics (T2) | Simple | 3.4 | 4.6 | 4.7 |
| Rate calculation via channels (T3) | Complex | 4.2 | 6.9 | 7.3 |
| Payment interface integration (T4) | Complex | 5.5 | 10.1 | 10.2 |
| Invoice processing (T5) | Very Complex | 8.5 | 12.5 | 13.7 |

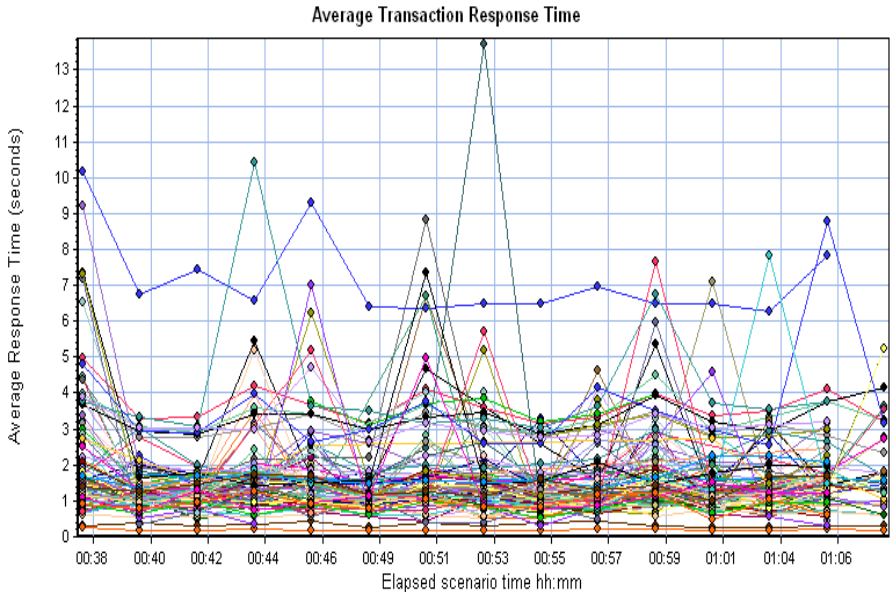


Fig. 2. Transaction Time


Fig. 3. Exploratory SLA vs Performance Test SLA

## VII. CONCLUSION

We want to conduct detailed complexity analysis on the modernized application and build some more unique exploratory test cases. This requires more data points by taking some more real time results into consideration. We want to apply the same pattern to hybrid cloud adoptions and also apply the approach on the public and private cloud deployment. As the modernized application is having huge data generation capabilities and we want to apply the exploratory testing approach to capture the efficiency on business insight modeling.

Thus we have proven that how an exploratory proactive model can help the enterprise modernized team to test and predict the SLA by not impacting the execution cycle or the business functionalities. As the enterprises are very keen on investment and more importantly more efficient in doing the modernization journey, the proposed approach will enable enterprise and the development team to efficiently modernize the applications. Hence the proactive exploratory testing approach is very important to achieve the desired results.

## REFERENCES


[1] B. Rochwerger, "The reservoir model and architecture for open federated cloud computing," IBM Journal of Research and Development, vol. 53, pp. 1-11, 2009.

[2] B. Urgaonkar, G. Paci_ci, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," Proceedings of the International Conference on Measurement and Modeling of Computer Systems ACM SIGMETRICS, Alberta, Canada, pp. 291-302, 2005.

[3] D. Petcu, "Multi-cloud: Expectations and current approaches," Proceedings of the international workshop on Multi-cloud applications and federated clouds (MultiCloud '13), NY, US, pp. 1-6, 2013.

[4] D. Stott, B. Floering, D. Burke, Z. Kalbarczyk, and R. Iyer, "Nftape: A framework for assessing dependability in distributed systems with lightweight fault injectors," in Proc. IPDS. IEEE, 2000, pp. 91–100.

[5] G. Kanawati, A. Kanawati, and J. Abraham, "Ferrari: A flexible software-based fault and error injection system," IEEE Transactions on

[6] Computers, vol. 44, no. 2, pp. 248–260, 1995.

[7] J. Bi, Z. Zhu, R. Tian, and Q. Want, "Dynamic provisioning modeling for virtualized multitier applications in cloud data center," Proceedings of the 3rd International Conference on Cloud Computing, Miami, Florida, US, pp. 370-377, 2010.

[8] J. Carreira, H. Madeira, and J. Silva, "Xception: Software fault injection and monitoring in processor functional units," Dependable Computing and Fault Tolerant Systems, vol. 10, pp. 245–266, 1998.

[9] K. Gao, Q. Wang, L. Xi, "Reduct algorithm based execution times prediction in knowledge discovery cloud computing environment," International Arab Journal of Information Technology (IAJIT), vol. 11, no.3, 2014.

[10] M.C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," Computer, vol. 30, no. 4, pp. 75–82, 1997.

[11] N. Grozev, and R. Buyya, "Multi-cloud provisioning and load distribution for three-tier applications," ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2014.

[12] S. Dawson, F. F Jahanian, and T. Mitton, "Orchestra: A probing and fault injection environment for testing protocol implementations," in Proc. IPDS. IEEE, 1996, p. 56. http://www.armored-computing.com/nftapeoverview.html.

[13] S. Ijaz, E. Munir, W. Anwar, and W.Nasir, "Efficient scheduling strategy for task graphs in heterogeneous computing environment", International Arab Journal of Information Technology (IAJIT), vol. 10, no. 5, 2013.

[14] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," Proceedings of the 4th International Conference on Autonomic Computing (ICAC 2007), Florida, US, 2007.

[15] http://www.infosys.com/engineering-services/serviceofferings/Pages/software-cloud-mobile-enablement.aspx.

[16] http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html.

[17] http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html.

[18] http://www.castsoftware.com/products/application-intelligenceplatform.

[19] http://jmeter.apache.org/.


## AUTHOR PROFILE


Ravikumar Ramadoss received his B.Sc., and MCA from Bharathidasan university, Trichy. He is doing Ph.D. (Computer Applications) in Sathyabama University, Chennai. He has over 14 years of software industry experience. He is working as Senior Technology Architect (STA) with Infosys, Bangalore. His areas of interest are Enterprise Modernization, Cloud ecosystems, Big Data adoptions and Proactive Performance Engineering. He holds various technology certifications in the field of J2EE and Cloud administrations.

Dr.N.M.Elango holds Ph.D in Computer Applications from SASTRA university, Thanjavur. He has over 28 years of experience in research and teaching. His areas of interest are image processing, enterprise modernization and machine learning. Having published papers in many international conferences and refereed journals of repute, he is the director of computer applications department with R.M.K engineering college and mentors research students in various fields of IT. He is a well-known academician and researcher in the academic and software industry.