Design Optimized Framework for Migrating a Legacy System to Cloud Environment

C.Udaya Shankar, P.T.Ajith, N.Naveen Kumar, S.Pushpalatha

***Abstract*— Cloud computing have attracted more and more enterprises to migrate their legacy applications to the cloud environment because of its high-scalability, on-demand nature. Although the cloud platform itself promises high reliability, ensuring high quality of service and high reliability is still one of the major concerns, since the enterprise applications usually consist of a large number of distributed components. Thus, improving the reliability of an application during cloud migration is a challenging and critical research problem. To address this problem, we propose a reliability-based Reliability Optimization Cloud (ROCloud) and consider all the constraint factors such as cost, etc., to improve the application reliability by fault tolerance. ROCloud includes two ranking algorithms. The first algorithm performs ranking of all the components, for an application, that will be migrated to the cloud. The second algorithm ranks components for hybrid applications that only part of their components are migrated to the cloud. Both algorithms employ the application structure information as well as the historical reliability information for component ranking. Based on the ranking result, optimal fault-tolerant strategy will be selected automatically for the most significant components with respect to their predefined constraints. The experimental results show that the reliability of the application can be greatly improved by refactoring a small number of error-prone components and tolerating faults of the most significant components**

***Index Terms*— Cloud migration, component ranking, fault tolerance, software reliability**

I. INTRODUCTION

Cloud computing enables convenient, on-demand network access to a shared pool of configurable computing resources. In the cloud computing environment, the computing resources (e.g., networks, servers, storage, etc.) can be provisioned to users on-demand. Startup companies can deploy their newly developed Internet services to the cloud without the concern of upfront capital or operator expense [5]. However, cloud computing is not only for startups, its cost effective, high scalability and high reliability features also attracted enterprises to migrate their legacy applications to the cloud [23]. Before the migration, enterprises usually have the concern to keep or improve the application reliability in the cloud environment. Thus, reliability-based optimization when migrating legacy applications to the cloud environment is becoming an urgently required research problem.

Manuscript received February 04, 2015.

C.Udaya Shankar, Department of Information Technology, Jeppiaar Engineering College,Chennai-119.

P.T.Ajith, Department of Information Technology, Jeppiaar Engineering College, Chennai-119.

N.Naveen Kumar, Department of Information Technology, Jeppiaar Engineering College, Chennai-119.

S.Pushpalatha, Assistant Professor, Department of Information Technology, Jeppiaar Engineering College, Chennai-119.

There are four major approaches in traditional software reliability engineering to improve system reliability: fault prevention, fault removal, fault tolerance, and fault forecasting.

Since the applications deployed in the cloud are usually complicated and consist of a large number of components, only employing techniques for fault prevention and fault removal are not sufficient. Another approach for building reliable systems is software fault tolerance, which is to employ functionally equivalent components to tolerate faults.

Software fault tolerance approach takes advantage of the redundant resources in the cloud environment, and makes the system more robust by masking faults instead of removing them.

Although the cloud platform is flexible and can provide resources on-demand, there is still a charge for using the cloud components (e.g., the virtual machines of Amazon Elastic Compute Cloud or Simple Storage Service). It will be expensive to provide redundancies for each component, since legacy applications usually involve a large number of components. To reduce the cost so as to assure highly reliability in a limited budget during the migration of legacy applications to cloud, an efficient reliability-based optimization framework is needed. Compared with newly developed applications, the reliability-based optimization of legacy applications has the following difficulties:

A. The failure rate of different components in a legacy application can vary. For example, some components in the legacy application are implemented by out-dated technology and have not been well maintained. These components can have great impact on application reliability. But they may not be selected as significant component by FTCloud, since FTCloud only employs structure information and does not take component failure rate information into consideration.
B. FTCloud needs expert knowledge to manually designate critical components. However, the migration team may not be the creator of the legacy application. So it will be difficult for them to manually list the critical components. Furthermore, the number of legacy applications as well as the number of components in these applications is large; it is thus impractical to manually identify critical components.
C.
D. Some applications may be restricted by enterprise security polices and only part of their components can be migrated to the cloud. Algorithms shall take these into account.

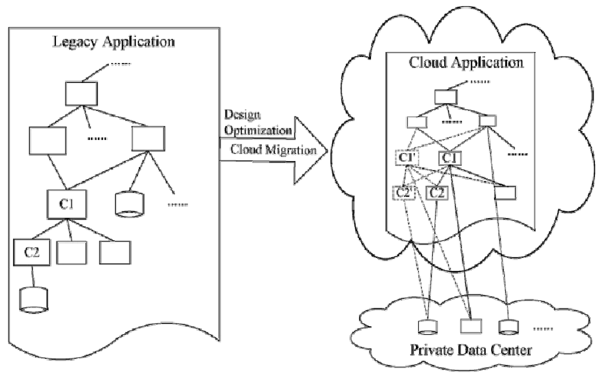

Fig. 1. Cloud migration example.

For these two reasons, FTCloud is not sufficient for improving the reliability of legacy applications. We need to take advantage of all materials of the legacy applications at hand, such as application logs, source code, etc. to automatically identify the components whose failures have great impact on the application reliability. Then provide backups for them using redundant resources in the cloud to improve the application reliability.

Based on this idea, we proposed Reliability-based Optimization in Cloud environment (ROCloud), which is a component ranking framework based on historical information to identify the significant components that have great impact on application reliability, and suggest optimal fault tolerance strategies automatically. ROCloud can help the designer optimize legacy application design to get a more reliable and robust cloud application effectively and efficiently.

The contribution of this paper includes:

This paper presents a design optimization frame-work for the cloud migration, named ROCloud. The main idea of this framework is first to identify significant components whose failures can have great impact on application reliability based on the application structure information and components reliability properties, and then provide fault-tolerant mechanism for these components to improve application reliability.

a) ROCloud includes two ranking algorithms. The first algorithm ranks components for the applications that all their components can be migrated to the cloud. The second algorithm ranks components for hybrid applications that only part of their components can be migrated to the cloud.

b) We conduct extensive experiments to evaluate the impact of significant components and their reliability properties on the reliability of the migrated application using reliability information of real-world Web services.

The rest of this paper is organized as follows. Section 2 lists the optimization challenges in cloud migration and proposes a three phase framework. Section 3 illustrates the details of the optimization framework. Section 4 concludes the paper, Section 5 includes references.

II. OPTIMIZATION FRAMEWORK FOR CLOUD MIGRATION

A. Optimization Challenges in Cloud Migration

First, we use a motivating example to show the challenging problems of this paper. Enterprise A wants to reduce upfront capital investment and system infrastructure maintenance effort. The cloud computing technology satisfies these requirements. Enterprise A decides to migrate its legacy applications to an IaaS cloud, as shown in Fig. 1. The legacy application consists of a number of distributed components. Ensuring reliability of the application is one of the major concerns for making the migration.

To enhance the system reliability, the designer wants to optimize the original design of legacy application by providing fault tolerance mechanisms for its components with replication techniques. When designing fault tolerance mechanisms for the components, the designer needs to consider the following problems:

a) Some components of the legacy application may be implemented by outdated technology and suffer from high failure rates. These components can have great impact on system reliability. Replication techniques are not enough to improve the reliability. For example, providing one replication for a component with failure rate 50 percent can only reduce the failure rate to 25 percent which is still unacceptable. A better approach is refactoring, that is to adopt new technology to rewrite the component and add fault prevention logic (e.g., exception handling), which can dramatically reduce the component's failure rate. Trade-offs need to be made when considering which components should be re-factored due to cost constraints.

b) The legacy application may consist of a large number of components. It is too expensive to deploy alternative replicas for all the components, since there are costs for using cloud resources (e.g., the virtual machines). To make trade-offs between costs and reliability, the designer chooses to tolerate faults of the most important components, whose failures have great impact on the whole system. However, it is not easy to identify which components have greater impact on system reliability, because:

- The reliability properties of each component may be very different. Some components may already have fault prevention logic (e.g., error checking, exception handling, etc.) and thus are more reliable than others.

- Failures of different components can have different impacts on the system. Components fulfilling critical tasks (e.g., payment) are taken as critical components, while other components accomplishing non-critical tasks (e.g., providing decorative pictures on web pages) are taken as non-critical ones [48]. Failures of critical components have greater impact on the system than failures of non-critical components.

These two characteristics should be considered in combination. A failure-prone non-critical component may have little impact on overall system reliability, while a component for critical task may be carefully designed and already have low enough failure rate. The straightforward approach to only consider components with high failure rates or fulfilling critical tasks as important components may not lead to an optimal solution.

c) Some applications are restricted by enterprise security polices and only part of their components can be migrated to the cloud. For these hybrid applications, the components which are kept in the private data center are potentially important components and they can only use resources in the private data center for fault tolerance.

d) There are a number of fault tolerance strategies. The cloud platform itself may also provide recovery approaches such as virtual machine restart. Different strategies have different overheads and costs. It is a challenging task for the designer to find out the optimal fault tolerance strategies for the significant cloud components.

To address the above problems, we first analyze the legacy application to collect the reliability properties and application structure information. Then, we proposes two significant component ranking algorithms in Section 3.2. At last an optimal fault tolerance strategy selection algorithm is presented in Section 3.3.2, which suggests optimal fault tolerance strategies for components with different constraints.

B. Optimization Framework

Fig. 2 shows the overview of our reliability optimization framework (named ROCloud), which includes three phases: 1) legacy application analysis; 2) automated significance ranking; and 3) fault tolerance strategy selection. The processes of each phase are as follows:

a) Both structure and failure information are extracted during the legacy application analysis phase. The structure information extraction consists of two sub-processes: component extraction and invocation extraction. The failure information including failure

rate and failure impact are collected from the execution logs and test results of the legacy application. Components with a failure rate higher than the threshold will be re-factored, and their reliability properties will be updated. A component graph is built for the legacy application based on the structure as well as the failure information.

b) In the automated significance ranking phase, two algorithms are proposed for ordinary applications that can be migrated to public cloud and hybrid applications that need to be migrated to hybrid cloud, respectively.

III. APPROACH

Optimization Framework approach has three modules:

a) Legacy application analysis,

b) Automated significance ranking,

c) Fault tolerance strategy selection.

The processes of each phase are as follows:

a) Both structure and failure information is extracted during the legacy application analysis phase. The structure information extraction consists of two sub processes: component extraction and invocation extraction. The failure information including failure rate and failure impact are collected from the execution logs and test results of the legacy application. Components with failure rates higher than the threshold will be re-factored and reliability properties are updated. A component graph is built for the legacy application based on the structure as well as the failure information.

b) In the automated significance ranking phase, two algorithms are proposed for ordinary applications that can be migrated to public cloud and hybrid applications that need to be migrated to hybrid cloud, respectively.

c) The performance, overhead, and cost of various fault tolerance strategy candidates are analyzed and the most suitable fault tolerance strategy is selected for each significant component based on its predefined constraint.

A. Legacy Application Analysis

The structure information includes components and the invocation information. The components are extracted from legacy applications by source code and documentation analysis. The invocation information such as invocation links and invocation frequencies can be identified from application trace logs. Source codes and documentations are useful supplementary materials in addition to trace logs. All the information are represented in a component graph. In this paper, the main optimization goal is reliability, so a more straightforward way is employed to determine which components should be re-implemented: components with failure rates greater than a threshold. The selection of the threshold is dependent on project budget and the target application failure rate.

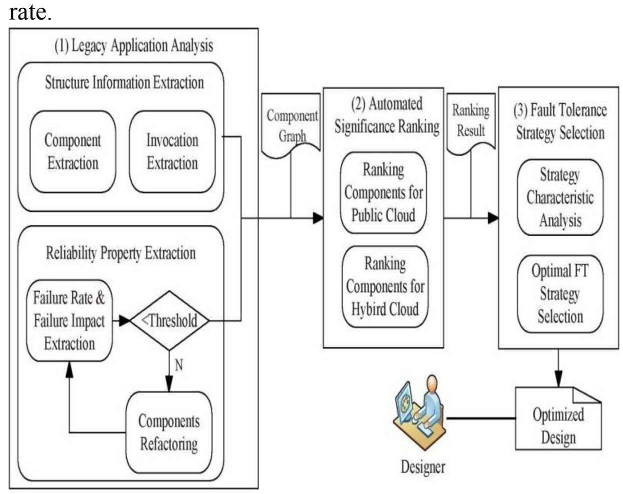

Fig. 2. Overview of the optimization framework

After refactoring, the component failure rates will be estimated based on test results, and the component reliability property dataset will be updated.

B. Automated Significance Ranking

Based on the component graph, two component ranking algorithms are proposed in this section. The first algorithm ranks components for ordinary applications where all their components can be migrated to the cloud. The second algorithm rank components for hybrid applications which can be partly moved to the cloud. In a distributed application, the failures of the components which are frequently invoked by many other components tend to have greater impact on the system compared with the components which are rarely invoked by others.

Thus these components are considered to be more important from the reliability aspect and should be ranked at the front of component list. Inspired by the Page Rank algorithm, we propose an algorithm to calculate the significance value of each component of the migratory application employing the component invocation relationships and reliability properties.

Based on the component graph and component reliability information, the component ranking algorithm includes the following steps:

1. Initialize by randomly assigning a numerical value between 0 and 1 to each component in the component graph.

2. Compute the significance value for a component c_i by:

$$V(c_i) = \frac{1-d}{n} f(c_i) p(c_i) + d \sum_{k \in N(c_i)} V(c_k) w_{ki},$$

With the above approach, the significance values of the components can be calculated by considering the application structure information, the invocation relationships, and the knowledge of component reliability properties in combination. A component with a larger significant value is considered to be more significant. The failures of these significant components will have great impact on other components and thus tend to cause application failures.

C. Fault Tolerance Strategy Selection

Software fault tolerance is widely adopted for critical systems (e.g., airplane flight control systems, nuclear power station management systems, etc.). At the same time, a cloud platform also provides approaches such as virtual machine restart, virtual machine migration, etc. to improve components reliability. By employing these techniques to provide functionally equivalent components, the component failures can be tolerated and thus the overall system reliability can be increased. Three well known software fault tolerance strategies as well as the approaches taking advantage of cloud platform features are introduced in the following with formulas for calculating the failure rate, response time and resource cost.

CONCLUSION

This paper presents a reliability-based design optimization framework for migrating legacy applications to the cloud environment. The framework consists of three parts: legacy application analysis, significant component ranking and automatic optimal fault-tolerant strategy selection. Two algorithms are proposed in the ranking phase: the first ranks components for the applications where all the components can be migrated to the cloud; the second ranks components for the applications where only part of the components can be migrated to the cloud. In both algorithms, the significance value of each component is calculated based on the application structure, component invocation relationships, component failure rates, and failure impacts. A higher significance value means the component imposes higher impact on the application reliability than others. After finding the most significant components, an optimal fault-tolerant strategy can be selected automatically with respect to the time and cost constraints.

In ROCloud, each component is considered as independent and the fault-tolerant strategy selection is carried out on component basis. In the future, we will study the fault tolerance of interrelated components. In addition, ROCloud uses the ratios of component failure to application failure to measure the failure impact of components. While the relationship between component failures and application failures can be complicated, more sophisticated models (e.g., Markov models, fault trees, etc.) will be investigated in the future work.

Our future work also includes:

1. Considering more factors (such as data transfer, invocation latency, etc.) when computing the weights of invocations links.
2. Taking the constraint factors such as cost into consideration during the ranking phase, and letting the designer know intuitively which components can make the biggest improvement while cost the least.

More experimental analysis on ROCloud and the impact of in-correct prior knowledge such as invocation frequen-cies and component failure rates.

REFERENCES

[1] S. Al-kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, ''VMFlock: Virtual Machine Co-Migration for the Cloud,'' in Proc. 20th Int. Symp. High Perform. Distrib. Comput., New York, NY, USA, 2011, pp. 159-170.

[2] A.A. Almonaies, J.R. Cordy, and T.R. Dean, ''Legacy System Evolution Towards Service-Oriented Architecture,'' in Proc. Int. Workshop SOAME, Madrid, Spain, Mar. 2001, pp. 53-62.

[3] G. Anthes. (). Security in the Cloud. Commun. ACM [Online].

[4] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G.Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, ''A View of Cloud Computing,'' Commun. ACM, vol. 53, no. 4, pp. 50-58, 2010.

[5] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G.Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, ''Above the Clouds: A Berkeley View of Cloud Computing,'' EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. EECS-2009-28, 2009.

[6] A. Avizienis, ''The Methodology of N-Version Programming,'' in Software Fault Tolerance, M.R. Lyu, Ed. Chichester, U.K.: Wiley, 1995, pp. 23-46.

[7] V. Batagelj and A. Mrvar, ''PajekVPajek: Analysis and Visualization

of Large Networks,'' Graph Drawing Softw., vol. 21, pp. 47-57, 2003.

[8] N. Bonvin, T.G. Papaioannou, and K. Aberer, ''A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage,'' in Proc. 1st ACM Symp. Cloud Comput., ser. SoCC'10, New York, NY, USA, 2010, pp. 205-216.

[9] S. Brin and L. Page, ''The Anatomy of a Large-Scale Hypertextual Web Search Engine,'' in Proc. 7th Int'l Conf. WWW, 1998, pp. 1-20.

[10] C. Cachin, I. Keidar, and A. Shraer. (). Trusting the Cloud. SIGACT News [Online]. 40(2), pp. 81-86. Available: http://doi. acm.org/10.1145/1556154.1556173

[11] M. Creeger, ''Cloud Computing: An Overview,'' ACM Queue, vol. 7, no. 5, pp. 1-5, June 2009.

[12] A.P.S. de Moura, Y.-C. Lai, and A.E. Motter, ''Signatures of Small-World and Scale-Free Properties in Large Computer Programs,'' Phys. Rev. E, vol. 68, p. 017102, 2003.

[13] J. Dean and S. Ghemawat, ''Mapreduce: Simplified Data Processing on Large Clusters,'' Commun. ACM, vol. 51, no. 1, pp. 107-113, Jan. 2008.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman,
A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, ''Dynamo: Amazon's Highly Available Key-Value Store,'' in Proc. 21st ACM SIGOPS Symp. Oper. Syst. Principles, ser. SOSP '07, New York, NY, USA, 2007, pp. 205-220.

[15] C.-L. Fang, D. Liang, F. Lin, and C.-C. Lin, ''Fault Tolerant Web Services,'' J. Syst. Architure, vol. 53, no. 1, pp. 21-38, 2007.

[16] S.S. Gokhale and K.S. Trivedi, ''Reliability Prediction and Sensitivity Analysis Based on Software Architecture,'' in Proc. ISSRE, 2002, pp. 64-78.

[17] F. Hao, T.V. Lakshman, S. Mukherjee, and H. Song, ''Enhancing Dynamic Cloud-Based Services Using Network Virtualization,'' in Proc. 1st ACM Workshop Virtualized Infrastruct. Syst. Archit., ser. VISA'09, New York, NY, USA, 2009, pp. 37-44.

[18] D. Hyland-Wood, D. Carrington, and Y. Kaplan, ''Scale-Free Nature of Java Software Package, Class and Method Collaboration Graphs,'' in Proc. 5th Int'l Symp. Empirical Softw. Eng., 2005, pp. 439-446.

[19] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, ''Ranking Significance of Software Components Based on Use Relations,'' IEEE Trans. Softw. Eng., vol. 31, pp. 213-225, Mar. 2005.

[20] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and
A. Goldberg, ''Quincy: Fair Scheduling for Distributed Computing Clusters,'' in Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Principles, ser. SOSP '09, New York, NY, USA, 2009, pp. 261-276. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629601

[21] F. Kamoun, ''Virtualizing the Datacenter Without Compromis-ing Server Performance,'' Ubiquity, vol. 2009, p. 2, Aug. 2009.

[22] A. Kertesz, G. Kecskemeti, and I. Brandic, ''An Sla-Based Resource Virtualization Approach for On-Demand Service Provi-sion,'' in Proc. 3rd Int. Workshop Virtualization Technol. Distrib. Comput., ser. VTDC'09, New York, NY, USA, 2009, pp. 27-34.

[23] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, ''Cloud Migration: A Case Study of Migrating an Enterprise IT Sstem to IaaS,'' in Proc. IEEE 3rd Int. Conf. CLOUD, 2011, pp. 450-457.

[24] K. Kim and H. Welch, ''Distributed Execution of Recovery Blocks: An Approach for Uniform Treatment of Hardware and Software Faults in Real-Time Applications,'' IEEE Trans. Com-put., vol. 38, no. 5, pp. 626-636, May 1989.

[25] H.A. Lagar-cavilla, J.A. Whitney, A. Scannell, P. Patchin, S.M. Rumble, E.D. Lara, M. Brudno, and M. Satyanarayanan, ''SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing,'' in Proc. 4th ACM Eur. Conf. Comput. Syst., New York, NY, USA, 2009, pp. 1-12.

[26] J. Laprie, J. Arlat, C. Beounes, and K. Kanoun, ''Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures,'' Computer, vol. 23, no. 7, pp. 39-51, July 1990.

[27] W. Li, J. He, Q. Ma, I.-L. Yen, F. Bastani, and R. Paul, ''A Framework to Support Survivable Web Services,'' in Proc. 19th IEEE Int'l Symp. Parallel Distrib. Process., 2005, p. 93.2.

[28] X. Lu, H. Wang, J. Wang, J. Xu, and D. Li, ''Internet-Based Vir-tual Computing Environment: Beyond the Datacenter as a Com-puter,'' Future Gener. Comput. Syst., vol. 29, no. 1, pp. 309-322, Jan. 2013.

[29] M.R. Lyu, ''Software Fault Tolerance,'' in Trends in Software. Hoboken, NJ, USA: Wiley, 1995.

[30] M.R. Lyu, Handbook of Software Reliability Engineering. New York, NY, USA: McGraw-Hill, 1996.

[31] P. Mell and T. Grance, ''The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technol-ogy,'' National Institute of Standards and Technology, Gaithers-burg, MD, USA, NIST Special Publication 800-145, 2011.

[32] M.G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, ''Thema: Byzantine-Fault-Tolerant Middleware Forweb-Service Applications,'' in Proc. 24th IEEE SRDS, 2005, pp. 131-142.

[33] D. Oppenheimer and D.A. Patterson, ''Studying and Using Failure Data from Large-Scale Internet Services,'' in Proc. 10th Workshop ACM SIGOPS Eur. Workshop, 2002, pp. 255-258.

[34] S.L. Pallemulle, H.D. Thorvaldsson, and K.J. Goldman, ''Byzan-tine Fault-Tolerant Web Services for N-Tier and Service Oriented Architectures,'' in Proc. 28th ICDCS, 2008, pp. 260-268.

[35] S. Pearson, ''Taking Account of Privacy When Designing Cloud Computing Services,'' in Proc. ICSE Workshop Softw. Eng. Challenges CLOUD, May 2009, pp. 44-52.

[36] B. Randell and J. Xu, ''The Evolution of the Recovery Block Concept,'' in Softw. Fault Tolerance, M.R. Lyu, Ed. Chichester, U.K.: Wiley, 1995, pp. 1-21.

[37] J. Salas, F. Perez-Sorrosal, N.-M. Marta Pati, and R. Jime´nez-Peris, ''Ws-Replication: A Framework for Highly Available Web Services,'' in Proc. 15th Int'l Conf. WWW, 2006, pp. 357-366.

[38] G.T. Santos, L.C. Lung, and C. Montez, ''FTWeb: A Fault Tolerant Infrastructure for Web Services,'' in Proc. 9th IEEE Int'l Conf. Enterprise Comput., 2005, pp. 95-105.

[39] G.-W. Sheu, Y.-S. Chang, D. Liang, S.-M. Yuan, and W. Lo, ''A Fault-Tolerant Object Service on Corba,'' in Proc. 17th ICDCS, 1997, pp. 393-366.

[40] S. Sivathanu, L. Liu, M. Yiduo, and X. Pu, ''Storage Management in Virtualized Cloud Environment,'' in Proc. 3rd IEEE Int'l Conf. Cloud, 2010.

[41] W.-T. Tsai, X. Zhou, Y. Chen, and X. Bai,
''On Testing and Evaluating Service-Oriented Software,'' IEEE Comput., vol. 41, no. 8, pp. 40-46, Aug. 2008.

[42] I. Ul Haq and E. Schikuta, ''Aggregation Patterns of Service Level Agreements,'' in Proc. 8th Int. Conf. Frontiers Inf. Technol., ser. FIT'10, New York, NY, USA, 2010, pp. 40:1-40:6.

[43] K.V. Vishwanath and N. Nagappan, ''Characterizing Cloud Computing Hardware Reliability,'' in Proc. 1st ACM Symp. Cloud Comput., ser. SoCC'10, New York, NY, USA, 2010, pp. 193-204. [Online]. Available: http://doi.acm.org/10.1145/1807128. 1807161

[44] S. White and P. Smyth, ''Algorithms for Estimating Relative Importance in Networks,'' in Proc. SIGKDD, 2003, pp. 266-275.

[45] S.M. Yacoub, B. Cukic, and H.H. Ammar, ''Scenario-Based Reliability Analysis of Component-Based Software,'' in Proc. ISSRE, 1999, pp. 22-31.

[46] W. Zhang, A. Berre, D. Roman, and H.
Huru, ''Migrating Legacy Applications to the Service Cloud,'' in Proc. 14th Conf. Companion. Object Oriented Programm. Syst. Languages Appl., ser. OOPSLA09, 2009, pp. 59-68.

[47] Z. Zheng and M.R. Lyu, ''A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services,'' in Proc. 6th ICWS, 2008, pp. 145-152.

Author Details:

C.Udaya Shankar
Department of Information Technology, Jeppiaar Engineering College,Chennai-119.

P.T.Ajith
Department of Information Technology, Jeppiaar Engineering College, Chennai-119.

N.Naveen Kumar
Department of Information Technology, Jeppiaar Engineering College, Chennai-119.

S.Pushpalatha
Assistant Professor,
Department of Information Technology, Jeppiaar Engineering College, Chennai-119.