International Journal of Pure and Applied Mathematics
Volume 117 No. 20 2017, 661-678
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

Method to Detect Dependent Component in Legacy Application for Cloud Migration

[1]T. Nandhini, [2]M. Hemalatha

[1]Research & Development Centre,

Bharathiar University,

Coimbatore, India.

Department of Computer Applications,

Indo-American College,

Cheyyar, India.

nandhini.t13@gmail.com

[2] Department of Computer Science,

Karunya University,

Coimbatore, India.

csresearchhema@gmail.com

Abstract

Cloudmigrationdealswithmovingtonewertechnologies,replaceretired products fullyorpartially. Deploying legacy applications on cloud involves several complicated issues and one of them is choosing the appropriate component. To achieve a cost-effective, reliable cloud migration focus has to be on the components which are to be deployed on the cloud. This paper deals with the importance of dependent components. An algorithm was proposed to deduce the dependent components of the main component based on the accessible and recurrent properties. The failure rate of dependent components increases the failure rate of main component which in turn increases the failure rate of application. The dependent components were ranked with the significant value. Fault tolerant strategies are employed on the dependent component and the experimental results show that the components with interrelated or dependent components show less failure rate and in turn application failure rate was alsolow. This paper mainly focuses on the selection of components based on the interdependency and shared components.

Key Words: Cloud migration, components, ranking, legacy application, interrelated.

1. Introduction

Cloud migration presents numerous challenges and raise security concerns, but it also allows an enterprise to potentially reduce capital expenditures and operating costs while also benefiting from the dynamic scaling, high availability, multi-tenancy. Enterprises that are built with legacy applications are interested in moving towards the newer technologies. The difficulties the enterprises experience form a broad area of research in migration. The old technology should be replaced or made to cope with the modern technology[1]. Migration process is a tedious process and should be handled carefully in order to benefit to the enterprise on cost factor. Cloud migration can be public, private or hybrid. The enterprise should decide well before migration, the plan to migrate the components of application[2].

Legacy application involves different technologies in it and to deploy that type of technology into the cloud is a challenging task and should be planned in advance before migration takesplace.Legacy applications were generally developed for high bandwidth, low-latency local area network(LAN) environments[3]. Legacy applications are often non-web in nature and have multiple tiers making them more technically complex than an organization's non-legacy application counterparts. Post migration of application has to survive in an environment in which new hardware technology would have been implemented [4].The structural analysis studyplays an important role while planning for migration.The developers of legacy application may have developed the security of the application based on the internal traffic but when it is deployed in cloud, the traffic is off-premise data centered.The most security oriented and confidential data should be kept in on-premise[5]. Main aim of migration is 1) to move to a sophisticated modern technology 2) to minimize the maintenance of the legacy enterprises 3) to achieve globalization of the legacy business. Cloud provides three main kinds of services [6] such as Infrastructure as a Service (Iaas) [7], Platform as a service (Paas) and Software as a Service (Saas).Critical and the non-criticalcomponents of a system [8][9]be analyzed using the structure analysis of the application and the critical components which are of more important for the application and more security oriented are kept in on-premise. Sophisticated model such as Markov model is appropriate for measuring the relationships between component failures and application failures. In Markov model, the state transitions are exponentially distributed, meaning that the transition rates are constant and therefore they are continuous[10]. Markov model assumptions are i) the transition probability depends on only the current state and not on the previous state,in this system is memory less,ii) transition probability are constant over time (system is stationary), iii) finite number of possible states are considered. Stochastic transition probabilityapproach to reliability modeling is appropriate for the circumstances in which the exact content and operation profile of the system not known in advance.

The rest of the paper is structured as follows, section2 presents related work, section3 presents reliable model in interrelated components, section 4 presents issues of component migration, section5 presents overview of dependent component framework and proposed solution, section6 presents simulation evaluation and evaluation results, section7concludes the paper.

2. Related Work

Component Ranking algorithms for legacy application migration in both public and private cloud were studied extensively in references [11] to [15]. References [11] and [12]proposed a fault tolerant cloud approach. In FTcloud approach [11], the significance value of a component is determined by the number of components that invoke this component and how often this component is invoked. An optimization selection algorithm was applied to provide fault tolerance strategies to the significant components.QOS driven component ranking framework was proposed in [12]. AccordingtoZheng, when designing a cloud application, the application designer has to select the optimal cloud componentfrom a set of functionally equivalent component candidates (i.e. $a_1 \ldots a_m$).In this paper ranking is done for QOS(Quality Of Service) based cloud services to build high quality cloud applications. In [13] a simulator to evaluate the fault tolerance policies was proposed.The simulator finds all the needed information via the schema such as 1) Application schema assume that only one application process or thread is executed on each node and the node count of an application remains static throughout its lifecycle. 2) Node schema stores the availability of the spare nodes.Two algorithms were proposed (i.e.) ROcloud1 and ROcloud2. The ROcloud1 algorithm deals with the situation in which all the components of the applicationsmigrated to the cloud. The ROcloud2 algorithm deals with the situation in which the enterprises are opting for hybrid model since some components may be critical and some may be non-critical, so the critical components are kept on-premise and the non-critical are migrated to the cloud. Kessel, Marcus, Atkinson, Colin proposed ranking software components for pragmatic reuse [14]. Reuse of software components done with three main activities such as selection, adaptation and integration. Software search engines and the tools to evaluate reuse components support were used in this paper. Metrics were applied in finding out the appropriate software components and to overcome the issues related was evaluated.Reinhardt D proposed ranking software components using a modified page rank algorithm includes safety aspects [15]. In this paper ranking is done for Qos based cloud services to build high quality cloud applications. Two algorithms was proposed as cloudrank1 and cloudrank2. The preference on a pair of cloud services was evaluated in cloudrank1.The values for calculating preferences could be explicit or implicit. The confidence level was evaluated using cloudrank2 algorithm. Dataset was based on a real world web services, more than 500 addresses was collected. Open source package in java was used for invocation code. Performance comparison was done with nine methods.

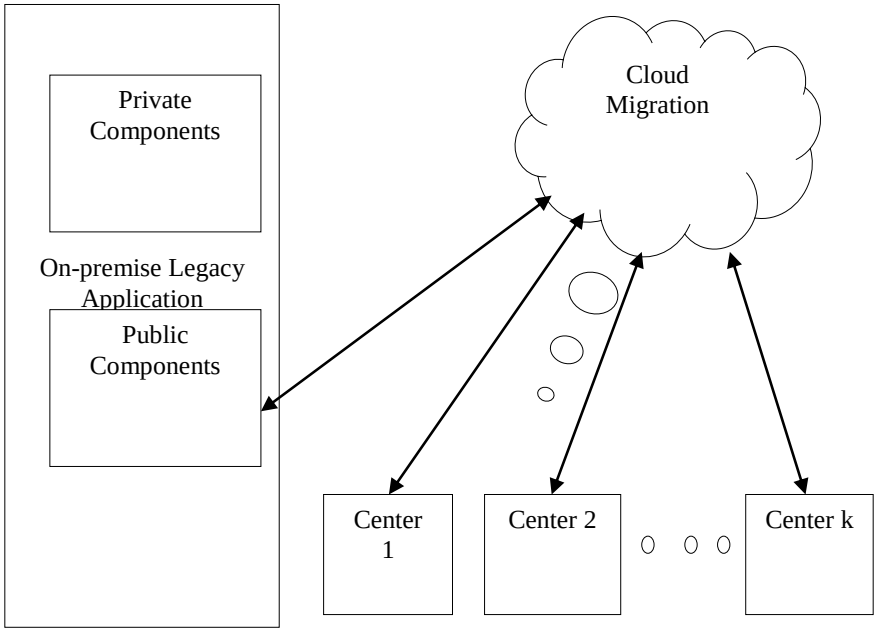

Figure 1: Architecture of Cloud Migration

3. Reliable Model For Interrelated Components

The reliable model for interrelated components proposed that the components of the legacy application are relying on the dependent components of the main component whose failure would cause the main component to fail and as a result the whole system fails. To overcome the failure of the main components, the dependent components should be given importanceand the failure rate of those components also should be taken into consideration.In this model the critical and the non-critical components of a system was found out using the structure analysis of the application and the critical components which are of more important for the application and more security oriented so those components are kept in on-premiseenterprise[16]. Less cost can be achieved only when correct decision is made for migration. Since the cloud environment is based on pay as you go basis,there arises a problem of finding out thecomponents that are moreconfidential and that are at theoperationlevel of applications.

In fig.1 private or critical components are kept in on-premise enterprise and the non-critical or the components which are considered for migration (public components) are migrated to the cloud.

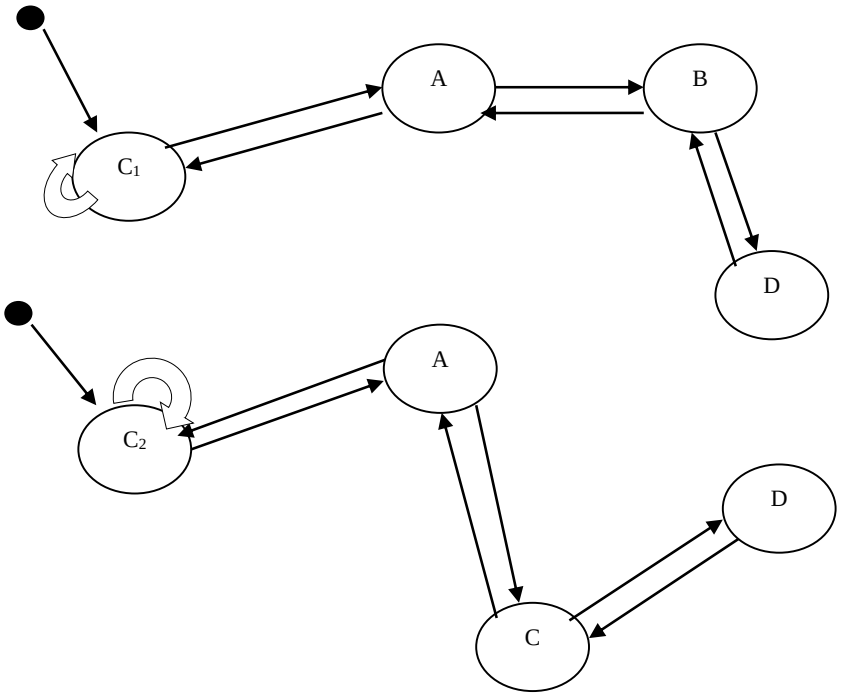

Figure 2: Dependent Components of C_1 and C_2

In fig.2 C1 and C2 are the main components and the main component C1 is assumed to be dependent on the components A, B and D. The main component C2 isassumed to be dependent onthe components A, C and D.Here the main component represents the component that was invoked first by the input given by the user.Components are considered as the functions which are invoked by an input from the user. To accomplish the task given to C1 it needs the result from the components A, B and D. Thus the component C1 is dependent on the components A, B and D.In the same way, the component C2 is dependent on the components A, C and D.The components A and D are shared by both C1 and C2. Thus, it has been emphasized that the components A and D should be ranked top so that the faulttolerance policies could be determined and could be adopted to overcome the failure of the main components C1 and C2.Since each component is described with a specific role to do, the fault tolerant policies of each component should also be separate.

4. Issues Of Component Migration

1. Legacy application consists of a number of distributed components.
2. Ensuring reliability, dependability, consistency and trustworthiness of the application is important during migration[17].
3. Some components are of outdated technology and so if they are migrated they would suffer from high failure rates.

4. These components have great impact on system reliability since legacy applications consists of large number of component [18][19], it is too expensive to replicate all the components.
5. The components have to be chosen for migration.

Cloud Computing has several benefits but it is vulnerable to large number of system failures and, as a consequence, there is an increasing concern among users regarding the reliability and availability of cloud computing services. Also, there is a need to implement autonomic fault tolerance technique for multiple instances of an application running on several virtual machines[20][21]. Server consolidation is an effective approach to maximize resource utilization while minimizing energy consumption in a cloud computing environment. Live VM(Virtual Machine) migration technology is often used to consolidate VMs residing on multiple under-utilized servers onto a single server, so that the remaining servers can be set to an energy-saving state. The major problem faced is of optimally consolidating servers in a data center. This is because server consolidation activities may hurt application performance. For server resources that are shared among VMs, such as bandwidth, memory cache and disk I/O, maximally consolidating a server may result in resource congestion when a VM changes its footprint (resource usage) on the server,thusaffecting the application performance[22].

5. Overview of Dependent Component Framework

Fig.3 showsoverview of dependent component framework. The components of the legacy application are extracted along with the number of invocations by structural analysis. The components thus extracted are classified into critical and non-critical. The critical component refers to the components which are to be retained in on-premise and the non-critical components are to be migrated to the cloud. The failure rates of all the components of the legacy application are calculated.The failure rates are calculated by the significant value equation and the components are ranked with the highest significant value first. The proposed interrelated component algorithm implemented on the extracted components, the dependent component and the shared components are extracted from it.Some of the dependent components are accessed by more than one main component then those components are treated as shared components in this framework. A dependent component graph was drawn based on the relationship between dependent component and the main component (the component which invoked the dependent component). Fault tolerance strategies are applied on the dependent components.

The failure rates of all the components before the extraction of dependent component and after the application of faulttolerance strategies to the dependent component was compared and found to be reduced.

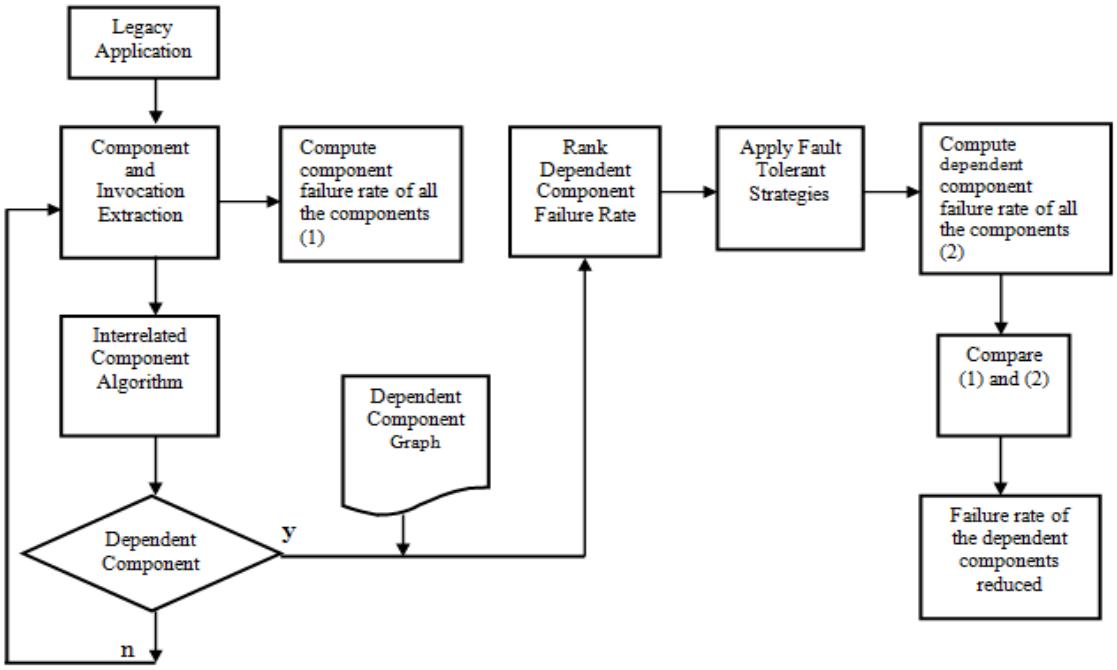

Figure 3: Overview of Dependent Component Framework

Fault Tolerance Strategies

Fault tolerance could be achieved through redundancy in hardware and software. Redundancy is simply the addition of information, resources or time beyond needed for normal system operation. Softwareredundancy is the addition of extra software, to perform a given function, to detect and tolerate faults [23]. The use of redundancy could provide additional capabilities for a system.

The popular approaches to detect software faults are

1. N-Version Programming (NVP).
2. Recovery Block (RB).

1. N-Version Programming (NVP)

NVP is a forward recovery scheme, it masks faults. NVP is a static technique. NVP relies on voting. Multiple versions of the same task are executed concurrently and the final result was determined by waiting until all n responses from theparallel invocations are reached. Its response time depends on the slowest version. The failure rate and response time for NVP[24] are calculated by

$$f = \sum_{i=(n+1)/2}^{n} F_i \quad (1)$$

$$t = max\, t_i \quad (2)$$

f is the failure rate, t is the response time,n is the number of redundant component,F_i is the failure rate of ithcomponent.

2. Recovery Block (RB)

Recovery block uses multiple alternates (backups) to perform the same function. Recovery block is a dynamic technique. It is a backward recovery scheme. The primary task executes first. When the primary task completes execution, its outcome is checked by an acceptance test. If the output is not acceptable, a secondary task undoing the effects of primary (i.e.,rolling back to the state at which primary was invoked) until either an acceptable output is obtained or the alternates are exhausted.The failure rate and response time for recovery block[25] are calculated by

$$f = \prod_{i=1}^{n} f_i \quad (3)$$

$$t = \sum_{i=1}^{n} t_i \prod_{k=1}^{i-1} f_i \quad (4)$$

f is the failure rate, t is the response time, n is the number of redundant component, f_i is the failure rate of i^{th}component.

Proposed Solution

1. Interrelated Component Algorithm

Inspired by the ROcloud2algorithm [26] and the Markov chain transition state properties[27] an algorithm was proposed to find the relationship between the dependent component failure rate and the invoked(main) component failure rate and to emphasize that the dependent components play a vital role on the application failure rate.

1. Based on the structural analysis and the invocation relationships of the migratory component a component dependent graph is drawn.
2. Initialize by assigning a random value between 0 and 1 to each component in the component graph.
3. When a component is invoked, the reachable rate to the other component is calculated by:

$$A_{ij}^{(nij)} > 0 \quad (5)$$

i and j are the states, n is the time required to move from one state to another.

4. When a component reached the last dependent component it has to return to the main component with the result, thus the return state is computed by

$$A_{jj} = 1 \quad (6)$$

$j$is the start and the end state of the components.

5. From step 3 and step 4 we can find out the list of the dependent components.By repeating the steps 3 and 4 for all the components, we can find out the shared components.
6. The significance value of the component can be calculated by:

$$V(c_i) = (1-d)\rho + d\sum_{k \in N(c_i)} V(c_k) wk_i \quad (7)$$

where

$$\rho = \begin{cases} 1/|p| & if\ c_i\ \varepsilon P \\ f(c_i)p(c_i)/|c| & if\ c_i\ \varepsilon C \end{cases}$$

N- number of components invoked by c_i , *P*- components of private data center , *C*- components of cloud, *d*- damping factor value is set as 0.85 to adjust the significance value derived from other components, w-weight of the edges, ρ represents the failure rate of i^{th}component $f(c_i)$ and the i^{th} component failure impact$p(c_i)$on the application to the total number of components of the application(componentson the cloud represented as *C*andon the private data center represented as *P*), $|P| + |C| = n$ where n is the total number of components of the application. The failure rate $f(c_i)$ can be calculated by

$$f(c_i) = \frac{\mu(c_i)}{\sum_{j=1}^{n} q\ ji} \tag{8}$$

$$p(c_i) = \begin{cases} \mu(a/c_i)/\mu(c_i) & if\ \mu(c_i) \neq 0 \\ 0 & if\ \mu(c_i) \neq 0 \end{cases} \tag{9}$$

$\mu(a/c_i)$ is the failure rate of application when component c_i fails.The more the significant value of a component, the component is more significant.

7. Fault tolerance strategies are applied for the dependent and shared components.

6. Simulation Evaluation

Case Study

A reporting application migration was considered as a case studyas in fig.4 to illustrate the process of interrelated component algorithm.

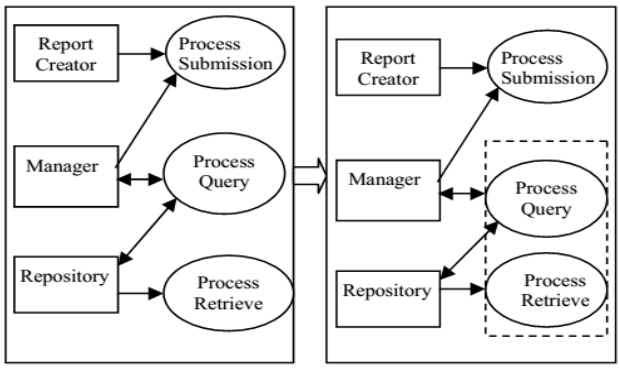

Figure 4: Case Study

Step 1 Identifying the Modules: First, from the selected legacy application and

its source code, the modules and the components related to each module of the application are identified.In this case, modules identified are report creator, report manager and report repository. Each module is associated with processes like report submission, query reports and retrieve reports as shown in fig.4.

Step 2 Ranking the components: Using the information of number of invocations for each component from the application log, the significance value of all the components are calculated by equation (7) and the most significant component are selected (Query report and retrieve report in this case) and are ranked [28] with the highest value first.

Step 3 Identifying the dependent component: The interrelated component algorithm are applied on the most significant components selected in step2 and the dependent components of the query report and retrieve report are identified and step2 is repeated to calculate the significance value of the dependent components. A dependent component graph is drawn.

Step 4 Optimal Fault Tolerance Strategy selection: Fault tolerance strategies are applied for dependent components ranked with top-k highest significant values. The failure rates of the main components with shared or dependent components are found to be reduced. The query report and the retrieve report are detected with shared and dependent components. These components are often invoked and thus results in high failure rate. Fault tolerant strategies applied on the dependent and shared components, resulted in very much less failure rates in main components.

Evaluation Model

The algorithm proposed in section 5.2.1 is evaluated. In the preliminary evaluation, the dependent component of C1 was found out (step3), after drawing a graph of dependent components and assigning random values to the edges (weight of the graph)[29].Page ranking was done by developing a structural signal using in-links, out-links and reachable values of the web pages of network graphs [30].

Experimental Setup

The effectiveness of our algorithm is evaluated with an experiment consisting of 5 PMs (Physical Machine) and 10 VMs (Virtual Machines). All PMs are configured with 4Gb RAM, 500Gb hard disk. The total number of VMsdefined for each datacenter here are maximum 40. Initially 10 VMs starting with basic configuration having 1 core CPU, 2Gb RAM, 160Gb hard disk were used.The experimental setup includes 200 components of real world applications. Call graphs for procedural code was used to extract the dependent components list. Each component failure rate is calculated by the ratio of the number of times failed to the number of times invoked. 200 sequences of random walk are generated so that each component would be tested. All the sequences are executed for 50 times and the corresponding results are shown in Table3.

Performance Comparison

The simulation setup explained above was employed to perform the four different methods NOFT, RB, NVP, ICCloud (Interrelated Component Cloud). In this, NOFT refers to no fault tolerance, RB refers to recovery block, NVP refers to N-version programming and ICCloud refers to interrelated component cloud.

Table1: Average Application Failure Rate

Node number	Methods	Threshold 0.01			Threshold 0.03			Threshold 0.05		
		10%	15%	20%	10%	15%	20%	10%	15%	20%
50	NOFT	0.02755	0.02667	0.02642	0.02606	0.02590	0.02540	0.02826	0.02816	0.02802
	RB	0.02409	0.02363	0.02311	0.02290	0.02250	0.02220	0.02800	0.02776	0.02750
	NVP	0.01942	0.01851	0.01733	0.01725	0.01709	0.01692	0.02700	0.02686	0.02528
	ICCloud	0.01089	0.01048	0.00971	0.00966	0.00944	0.00932	0.01084	0.01075	0.01064
100	NOFT	0.03082	0.03085	0.03080	0.02952	0.02926	0.02901	0.03034	0.03029	0.03026
	RB	0.02376	0.02251	0.02143	0.02392	0.02366	0.02344	0.02771	0.02660	0.02610
	NVP	0.02096	0.01977	0.01908	0.01992	0.01972	0.01988	0.02506	0.02472	0.02444
	ICCloud	0.01079	0.01022	0.00938	0.00928	0.00918	0.00902	0.01292	0.01266	0.01222
150	NOFT	0.02682	0.02677	0.02653	0.03164	0.03132	0.03101	0.03315	0.03345	0.03378
	RB	0.02358	0.02314	0.02309	0.02498	0.02371	0.02333	0.03078	0.03058	0.03011
	NVP	0.02212	0.01942	0.01918	0.01999	0.01971	0.01965	0.02888	0.02866	0.02842
	ICCloud	0.00961	0.00956	0.00944	0.00881	0.00868	0.00842	0.01178	0.01152	0.01146
200	NOFT	0.02576	0.02561	0.02499	0.03288	0.03271	0.03268	0.03445	0.03432	0.03411
	RB	0.02256	0.02115	0.02001	0.02538	0.02522	0.02511	0.03178	0.03156	0.03142
	NVP	0.01844	0.01644	0.01542	0.01672	0.01678	0.01682	0.02780	0.02756	0.02744
	ICCloud	0.00941	0.00930	0.00902	0.00606	0.00581	0.00566	0.00980	0.00972	0.00966

The number of times of invocations and the failure rates of the components are tested with the components in application (Node Numbers) by 50,100,150,200 and the results are shown in Table1. NOFT resulted in the worst performance because of no fault tolerance strategies were employed. The recovery block method has given better performance when compared to NOFT. NVP was better when compared to RB. ICCloudhasgiven the bestperformance when compared to all the other three methods.Thus, the ICCloud emphasized the role of dependent components.The failure rates of the main components was reduced

when the fault tolerance strategies was employed to the dependent or interrelated components.

The shared and dependent component would be considered for analyzing fault tolerance of the components. If these components have less failure rate, then the failure of onecomponent would not affect the other invoking components which are dependent on it.In this experiment, the methods RB, NVP are employed on the main component with the highest significant value. The components with the highest significance value whose failure rate also highestbecause of the unavailable components at the time of its need. Thus, the components are ranked with the highest significant value first and the top-k components are extracted. The ICCloud method was employed on the extracted list of components to detect the dependent component and shared components. The significant values of the extracted componentswere calculated using the equation (7). The result tabulated in Table1proved that the method ICCloudhas the lowest application failure rate.

Scalability of the Algorithm

The ICCloud algorithm's performances are measured with the factors like response time and failure rate of the components using equations (1) to (4) of fault tolerant strategies such as N-Version Programming and Recovery Block.The fig.5 and fig.6show the performance of the four methods with 200 components and the threshold value set as 0.01, 0.03 and 0.05. The ICCloud perform better when compared to other fault tolerant mechanisms.

The top-k value of the components was selected with the threshold values set as 0.01,0.03 and 0.05. With the threshold value set as 0.01, the failure rates of the application generated are listed with top-k values as 10%, 15% and 20%.Likewise the threshold set with 0.03 and 0.05 listed the application failure rates of the components and the top-k values of the components with 10%, 15% and 20% are tabulated.The results as tabulated describes that the increase in top-k value settings by 5% in each stage has given a failure rate less than the previous settings.The response time of the recovery block (RB) is greater when compared to n-version programming (NVP).The failure rate of recovery block was high when compared to NVP.

The recovery block invokes the standby components sequentially when the primary component fails and its response time is the summation of the execution time of all the failed versions and the first successful one.

The response time of the ICCloud method was found to be much better when compared to the other methods.TheICCloud algorithm performs with lesser response time with any of the selected fault tolerance mechanisms such as RB and NVP. Thus, it emphasized the importance of replication of dependent components, so that the main component would be able to access the shared components which results in less failure rate of the main component and smooth functioning of the application.The threshold value is necessary in this

experimentbecause all the components of the applicationwouldnot be 100% fault tolerant. The ICCloud algorithm illustrates the need for fault tolerance to the dependent component but not fully free from the faults, it would only bring down the failure rate. The threshold value was selected according to the failure rates of the individual components, the number of invocations and the number of components selected for the experiment.

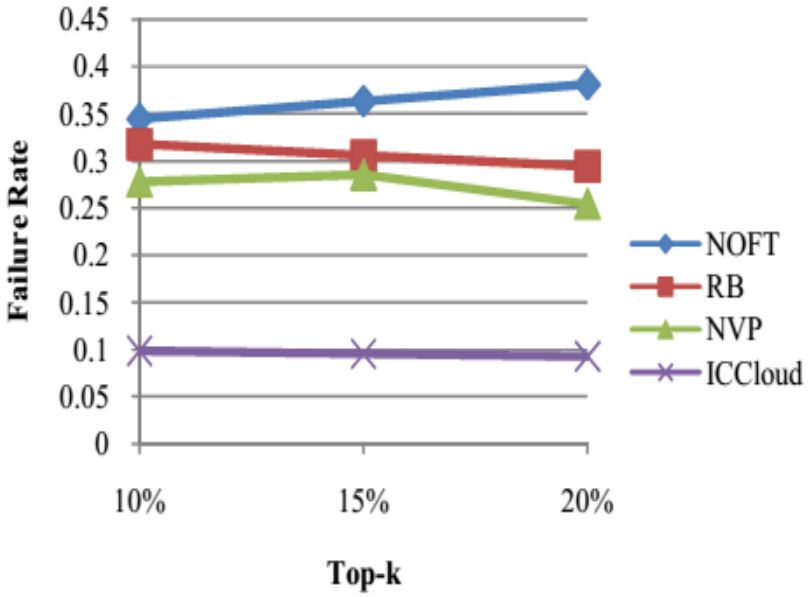

Figure 5

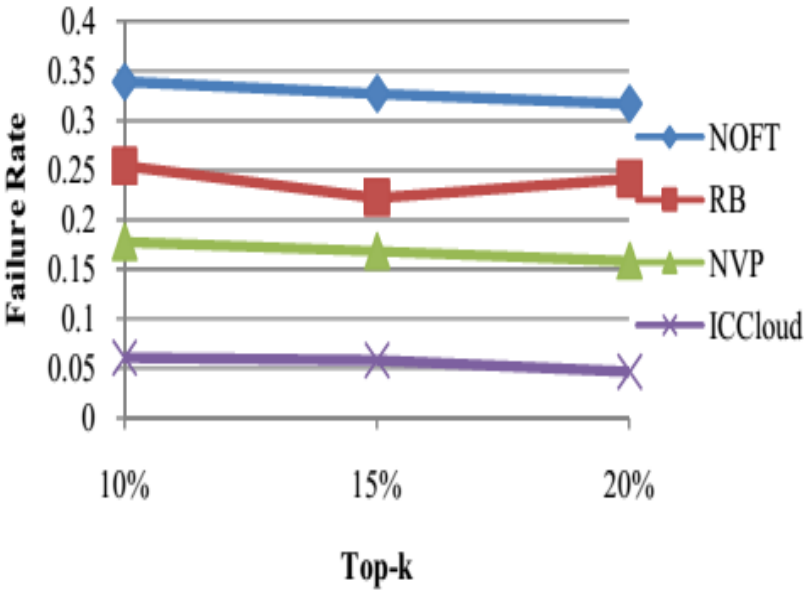

Figure 6

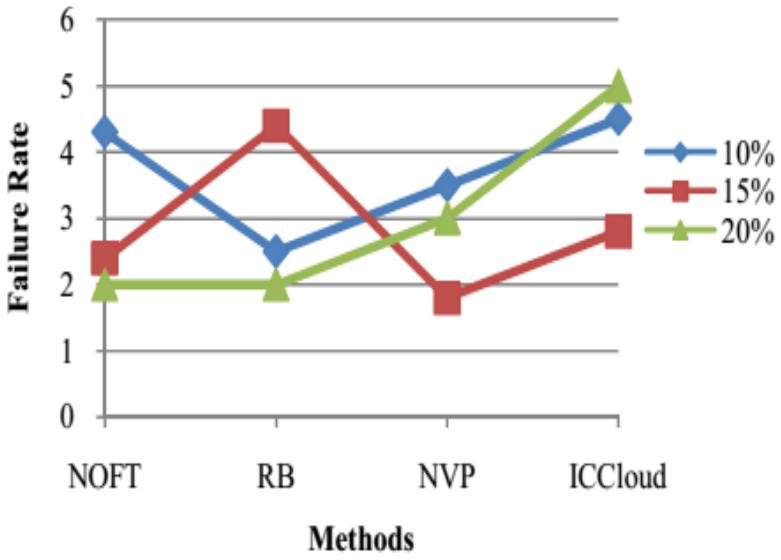

Figure 7

Impact of Dependent Component Failure on Application Failure Rate

The impact of dependent component failure on the application was studied by comparing the failure rate of the four methods NOFT, RB, NVP, ICCloud. The equation (5) was employed to calculate the dependent component failure impact. If equation (5) results in 1, then the dependentcomponent failure affected the performance of the main component which in turn affects the performance of the application. The average dependent component failure impact settings considered are 0.25, 0.5 and 1.

- The fig.7, show the performance of the four methods for 50 components and with the threshold values of 0.01, 0.03 and 0.05. The ICCloud method outperforms the usual fault tolerance mechanisms.
- The top-k values set as 10%,15% and 20% clearly indicated that the curve of ICCloud decreases with the increase of the number of components. Different threshold values were used for different top-k values and analyzed the performances of the methods.
- The fact that the more the component redundancy, affects the performance of the application was also considered and the threshold values and the top-k values of the components selected were only up to first 20% of the total components and the results show that the more the number of components the less was the fault rate occurred.
- The impact of shared component was very high thus the redundant components of the shared components brought down the high unavailability rate of the components and the main component show the lower failure rates at different threshold values.

7. Conclusion and Future Enhancement

The benefit of cloud migration of legacy application lies in the achievement of

reliability and cost minimization. The hybrid component migration approach is appropriate for the migration of components of legacy application. This paper emphasized that the software fault tolerance mechanism applied on the identified dependent components and the shared components of the application highly reduced the failure rate thus increasing the effect of reliability among the components. The selection of appropriate components for migration with appropriate fault tolerant mechanism minimizes the cost of migration. The experimental results described the failure rate of dependent component affected the failure rate of the application. The network delay and the impact of network delay on the dependent component migration will be for future study.

References

[1] Buyya R., Yeo C.S., Venugopal S., Broberg J., Brandic I., Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for delivering Computing as the Fifth Utility, Future Generation Computer Systems 25 (6) (2009), 599-616.

[2] Pahl C., Xiong H., Walshe R., A Comparison of On-Premise to Cloud Migration Approaches - A Tale of Four Cloud Migration Processes, Proc. European Conf. Service-Oriented and Cloud Computing, (2013).

[3] Pearl Brereton, Kitchenham B.A., David Budgen, Mark Turner, Mohamed Khalil, Lessons from applying the systematic literature review processwithin the software engineering domain, The Journal of Systems and Software (2006), 571–583.

[4] S¨oren Frey, Wilhelm Hasselbring, The CloudMIG Approach: Model-Based Migration of Software Systems toCloud-Optimized Applications, International Journal on Advances in Software 4 (3 & 4) (2011).

[5] Khadija Sabiri, Faouzia Benabbou, Methods of Migration from On-premise to Cloud, IOSR Journal of Computer Engineering (IOSR-JCE) 17 (2) (2015), 58-65.

[6] Xianrong Zheng, Patrick Martin, Kathryn Brohman, Li Da Xu, Cloud Qual: A Quality Model for Cloud Services, IEEE Transactions on Industrial Informatics 10 (2) (2014).

[7] Pieter Van Gorp, Paul Grefen, Supporting the Internet-based evaluation of research software with cloud infrastructure, Software System Model, Springerlink.com. 11 (2010), 11–28.

[8] Garlan D., Monroe R.T., Wile D. Acme, Architectural Description of Component-Based Systems,Foundations of Component-Based Systems, Leavens, G.T., and Sitaraman, M.(eds). Cambridge University Press (2000), 47-68.

[9] Jun-Feng Zhao, Jian-Tao Zhou, Strategies and Methods for Cloud Migration**,** International Journal of Automation and Computing (2014), 143-152.

[10] Puterman M.L., Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley Online Library 414 (2009).

[11] Zibin Zheng, Tom Chao Zhou, Michael R. Lyu, Irwin King, Component Ranking for Fault-Tolerant Cloud Applications, IEEE transactions on Services Computing 5 (4) (2012).

[12] Zheng Z., Zhang Y., Lyu M.R., CloudRank: A QoS-Driven Component Ranking Framework for Cloud Computing, IEEE Proc. Int'l SRDS (2010), 184-193.

[13] Zheng Z., Zhou T.C., Lyu M.R., King I., FTCloud: A Ranking-Based Framework for Fault Tolerant Cloud Applications, IEEE Proc. ISSRE (2010), 398-407.

[14] Kessel M., Atkinson C., Ranking software components for pragmatic reuse, In Proceedings of the Sixth International Workshop on Emerging Trends in Software Metrics (2015), 63-66.

[15] Reinhardt D., Ranking software components using a modified PageRank algorithm including safety aspects, In IEEE International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV) (2014), 274-281.

[16] Nigel Cook, Dejan Milojicic, Vanish Talwar, Cloud management, Journal of Internet Service Application 3 (2012), 67–75.

[17] Rashmi Rai, Gadadhar Sahoo, Shabana Mehfuz,Exploring the Factors influencing the cloud computing adoption: a systematic study on cloud Migration, Springer Plus Springer Open journal (2015).

[18] Krishnamurthy S., Mathur A.P., On the estimation of reliability of a software system using reliability of its components, Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE-97) (1997), 146-155.

[19] Plasil F., Visnovsky S., Behavior Protocols for Software Components, IEEE Transactions on Software Engineering 28 (11) (2002),1056–1076.

[20] Ravi Jhawar and Vincenzo Piuri, Fault Tolerance Management in IaaS Clouds, IEEE Transactions on Services Computing 4 (2012), 101-148.

[21] Ricky W.B., Sally C.J, Techniques for Modeling the Reliability of Fault-Tolerant Systems With the MarkovState-Space Approach, NASA Reference Publication, (1995).

[22] Bhardwaj S., Jain L., Jain S., Cloud computing: A study of infrastructure as a service(IAAS), International Journal of Engineering and Information Technology 2 (1) (2010), 60–63.

[23] Aniruddha G.S., Wael R.E., Samantha S.F., Byung H.P., David E.B., Randall B., Strategies for Fault Tolerance in Multicomponent Applications, International Conference on Computational Science, ICCS, Procedia computer science (2011), 2287-2296.

[24] Avizienis, The Methodology of N-Version Programming in Software Fault Tolerance, M.R. Lyu, Ed. Chichester, U.K. Wiley, (1995), 23-46.

[25] Kim K., Welch H., Distributed Execution of Recovery Blocks: An Approach for Uniform Treatment of Hardware and Software Faults in Real-Time Applications, IEEE Trans. Computers 38 (5) (1989), 626-636.

[26] Weiwei Qiu, Zibin Zheng, Xinyu Wang, Xiaohu Yang, Michael R. Lyu, Reliability-Based Design Optimization for Cloud Migration, IEEE Transactions on Services Computing 7 (2) (2014).

[27] Reussner R., Schmidt H., Poernomo I., Reliability prediction for component-based software architectures, Journal of Systems and Software, Elsevier Science Inc, 66(3) (2003), 241-252.

[28] Inoue K., Yokomori R., Yamamoto T., Matsushita M., Kusumoto S., Ranking Significance of Software Components Based on UseRelations, IEEE Transactions Software Engineering 31 (2005), 213-225.

[29] Hema Dubey, Roy B.N., An Improved Page Rank Algorithm based on Optimized Normalization Technique, International Journal of Computer Science and Information Technologies 2 (5) (2011), 2183-2188.

[30] Neelam Tyagi, Simple Sharma, Weighted Page Rank Algorithm Based on Number of Visits of Links of Web Page, International Journal of Soft Computing and Engineering (IJSCE) 2 (3) (2012).