TELKOMNIKA, Vol.14, No.4, December 2016, pp. 1565~1574
ISSN: 1693-6930, accredited A by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/TELKOMNIKA.v14i4.3964

DCR: Double Component Ranking for Building Reliable Cloud Applications

Lixing Xue, Zhan Zhang*, Decheng Zuo
School of Computer Science and Technology, Harbin Institute of Technology,
Harbin 150001, Heilongjiang Province, China
*Corresponding author, e-mail: zz@ftcl.hit.edu.cn

Abstract

Since cloud applications are usually large-scale, it is too expensive to enhance the reliability of all components for building highly reliable cloud applications. Therefore, we need to identify significant components which have great impact on the system reliability. FTCloud, an existing approach, ranks the components only considering the impact of component internal failures and ignoring error propagation. However, error propagation is also an important factor on the system reliability. To attack the problem, we propose an improved component ranking framework, named DCR, to identify significant components in cloud applications. DCR employs two individual algorithms to rank the components twice and determines a set of the most significant components based on the two ranking results. In addition, DCR does not require information of component invocation frequencies. Extensive experiments are provided to evaluate DCR and compare it with FTCloud. The experimental results show that DCR outperforms FTCloud in almost all cases.

Keywords*: Component Ranking, Cloud Application, System Reliability, Error Propagation, Internal Failure*

1. Introduction

Cloud computing is an Internet-based computing paradigm, which provides shared processing resources and data to computers and other devices on demand [1, 2]. In recent years, cloud computing is becoming more and more popular and many enterprises and individuals prefer to build their systems in the cloud environment. The software systems in the cloud are named as cloud applications which usually consist of various cloud components communicating with each other. The cloud applications are usually large-scale and very complex [3], which may pose a threat to the system reliability and hinder transferring critical systems to the cloud. Nowadays, end-users hate applications with low-reliability and the demand for high reliability is continually increasing. Building highly reliable cloud applications has become a challenging and required research problem.

The major approach for improving the cloud application reliability is to enhance the reliability of each individual component. This may be accomplished either by employing functionally equivalent but more reliable components to reduce component failures or by adding fault-tolerance strategies to tolerate component failures. Unfortunately, both of them will incur extra cost. As cloud applications usually involve a large number of components, it is too expensive to provide alternative components or add fault-tolerance strategies for all the components. Based on the 80-20 rule [4], FTCloud-an existing approach [5] attempts to improve the reliability of cloud applications by ranking the components to identify a small set of significant components and enhancing their reliability. However, FTCloud only considers the impact of the component internal failures on the system and does not take into account error propagation which is also a serious threat to the global reliability [6].

To attack the problem, we propose a component ranking framework for identifying significant components and helping designers to build highly reliable cloud applications. This framework includes two component ranking algorithms, taking into account the direct impact of component internal failures on the system and the harm of error propagation, respectively. Based on the two algorithms, two ranking results are obtained and then a small set of the most significant components which have great impact on the system reliability are determined. By enhancing the reliability of these significant components, the system reliability can be greatly

Received May 10, 2016; Revised October 22, 2016; Accepted November 8, 2016

improved. Due to the fact that the components are ranked twice, we name the framework DCR-double component ranking.

The main contribution of this paper is:

1. This paper identifies the importance of error propagation in locating significant components of cloud applications which is not considered by FTCloud, and proposes an improved component ranking framework, named DCR. DCR only employs component invocation relationships to independently rank the components twice and selects the critical components which have great impact on the system reliability from the two ranking results.
2. Extensive experiments are provided to evaluate the impact of significant components identified by DCR on the reliability of cloud applications and draw performance comparison between DCR and FTCloud. The results show that DCR is effective and outperforms FTCloud in almost all cases.

The rest of this paper is organized as follows. Section 2 introduces the two descriptions of significant components, the system architecture of DCR and related work. Section 3 details the double component ranking framework. Section 4 shows the experiments to evaluate DCR. Section 5 draws the conclusion and future work finally.

2. Preliminaries

2.1. Significant Components

A failure of a component in software systems can be attributed to two reasons [6], as shown in Figure 1. One is that an error caused by faults in the component (such as bugs) is delivered at the output interface, i.e. component internal failure. The other is that the component receives an incorrect input and generates an erroneous output, namely error propagation leads to a component failure. A system failure occurs only if an error eventually reaches the system interface, no matter how the error is produced and propagated. In a word, component internal failures and error propagation are two major threats to the system reliability.

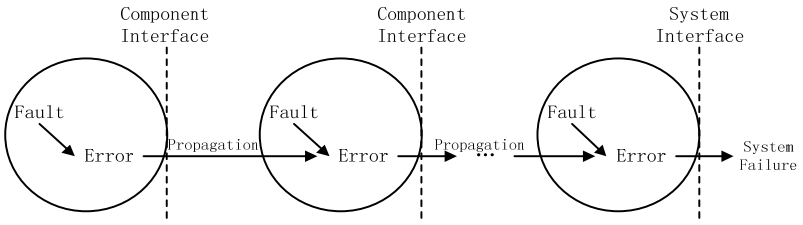

Figure 1. Two threats to reliability

It is apparent that not only the direct impact of component internal failures on the system should be reduced, but also the harm of error propagation should also be minimized, if we want to build highly reliable cloud applications. Therefore, the significant components in this paper are able to be described from two perspectives.

1· The significant components are the ones whose failures have great impact on the system.
2· The significant components are also the ones which may severely affect a lot of other components and further harm the global reliability by propagating errors out when they fail.

2.2. System Architecture

The system architecture of DCR is showed in Figure 2, which includes three parts: structure graph building, component ranking and significant component determination. The procedures of DCR are as follows:

1. The system designer provides the structure information of a cloud application to DCR. A structure graph is generated based on the component invocation relationships.
2. Two series of significance values of the cloud components are calculated by employing two different component ranking algorithms which are proposed in terms of the two descriptions of significant components in the last subsection, respectively. According to the two series of significance values, the components are ranked twice.

3. Based on the two ranking results, the most significant components which have strong impact on the global reliability are determined and returned to the system designer for building a reliable cloud application.

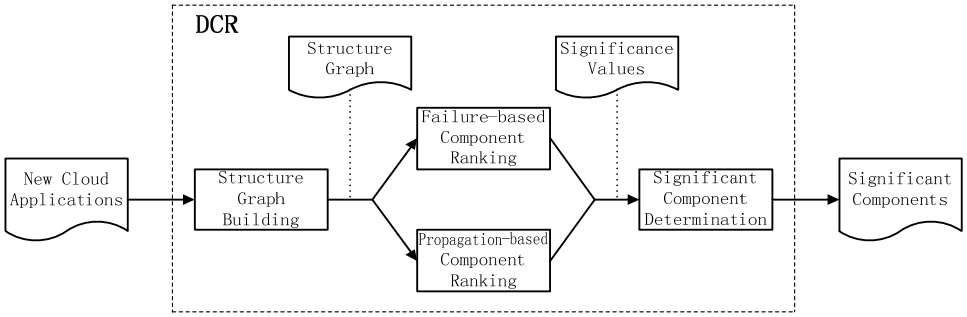

Figure 2. System architecture of DCR

2.3. Related Work

In traditional software reliability engineering, there are four common methods to build reliable software systems, namely fault prevention, fault removal, fault tolerance and fault forecasting [7]. However, fault prevention and fault removal are not able to be applied when we build cloud applications. This is because building cloud applications usually uses existing cloud components and we cannot participate in the development of them. But we can select components with high reliability according to design requirements. Another method we can employ is software fault tolerance. Software fault-tolerance techniques, such as recovery block [8] and N-Version Programming (N-Modular Redundancy) [9], are widely used in various systems. In the cloud environment, a great number of functionally equivalent but independently designed components can be used for designing fault-tolerance mechanisms.

As cloud computing is becoming popular, a number of works have been carried out on it. Service component selection and composition is one of the hotspots. Many approaches have been proposed, such as QoS-aware web service composition [10], web service reputation model [11], OWL-S service profile based web service selection [12] and web service selection based on concurrent requests [13]. Component ranking is a prerequisite for applying these research findings and some studies have been carried out. However, the approaches do not take into account error propagation, which is also a major threat to the reliability of cloud applications. In addition, they require the structure information as well as the information of component invocation frequencies. Our approach which attacks the weakness requires only the structure information and takes into account the harm of error propagation in the system, obtaining wonderful results.

3. Double Component Ranking

As shown in Figure 2, DCR includes three parts, which will be detailed in this section, respectively. Structure graph building is introduced within Section 3.1. Then the two component ranking algorithms are proposed according to the two descriptions of significant components in Section 3.2 and 3.3 respectively. In Section 3.4, determination of significant components is discussed.

3.1. Structure Graph Building

The structure of a cloud application, that is, the component invocation relationships, can be modeled as a directed graph $G =< C, E >$, where a node C_i in the node set C denotes a component and a directed edge e_{ij} from C_i to C_j in the edge set E represents that C_i invokes C_j (denoted as $C_i \rightarrow C_j$).

A structure graph containing n nodes can be described by an $n \times n$ adjacency matrix $A = \left(a_{ij}\right)_{n \times n}$. Each entry a_{ij} in the matrix is defined by:

$$a_{ij} = \begin{cases} 1 & if\ C_i \to C_j \\ 0 & else \end{cases} \quad (1)$$

In the matrix, $a_{ii} = 1$ represents a self-invocation of component C_i. The number of edges starting from node C_i is called out-degree of C_i, denoted as $\deg^-(C_i)$. It is able to be obtained by:

$$\deg^-(C_i) = \sum_{k=1}^{n} a_{ik} \quad (2)$$

Similarly, the number of edges ending at node C_i is called in-degree of C_i, denoted as $\deg^+(C_i)$. It can be calculated by:

$$\deg^+(C_i) = \sum_{k=1}^{n} a_{ki} \quad (3)$$

A component C_i invokes a total of $\deg^-(C_i)$ components and C_i is invoked by a total of $\deg^+(C_i)$ components.

3.2. Failure-Based Component Ranking

In a cloud application, some components are frequently invoked by many other components. It is obvious that their failures will directly affect the system reliability much more than other components [14]. These components follow the first description of significant components discussed in the last section. Intuitively, these significant components in a structure graph are the ones which have many ingoing links from other important components. On the basis of the PageRank algorithm [15], we propose an algorithm to calculate the first series of the significance values of the cloud components, named as failure-based significance values.

For a cloud application which contains n components, the failure-based significance value $VF(C_i)$ of a component C_i is defined as:

$$VF(C_i) = \alpha \sum_{C_j \to C_i} \frac{VF(C_j)}{\deg^-(C_j)} + (1-\alpha)\frac{1}{n} \quad (4)$$

Where $\frac{1}{n}$ is the basic significance value of C_i itself while $\sum_{C_j \to C_i} \frac{VF(C_j)}{\deg^-(C_j)}$ is the significance value derived from other components that invoke C_i. The parameter α $(0 \le \alpha \le 1)$ in (4) is utilized to adjust the proportion of the two values, which is usually set to be 0.85. By (4), a component C_i has a large failure-based significance value if the sum of failure-based significance values of the components that invoke C_i is large, indicating that C_i is invoked by many other significant components.

Equation (4) can be written in matrix form:

$$\begin{bmatrix} VF(C_1) \\ \vdots \\ VF(C_n) \end{bmatrix} = \alpha W \begin{bmatrix} VF(C_1) \\ \vdots \\ VF(C_n) \end{bmatrix} + (1-\alpha) \begin{bmatrix} 1/n \\ \vdots \\ 1/n \end{bmatrix} \quad (5)$$

Where the matrix $W = (w_{ij})_{n \times n}$ is defined by:

$$w_{ij} = \begin{cases} \dfrac{1}{\deg^{-}(C_j)} & if\ C_j \to C_i \\ 0 & else \end{cases} \tag{6}$$

The procedures of calculating the failure-based significance values are simple. First, randomly assign initial values between 0 and 1 to the failure-based significance values $VF(C_i)$ ($i = 1, 2 \cdots n$). Then, solve (5) by repeating the computation until all significance values become stable.

Using the above approach, the failure-based significance values of the cloud components can be obtained. According to these values, the components are ranked. A component with a larger value is considered to be more significant. The failures of the significant components selected from this ranking result will have great impact on the system reliability.

3.3. Propagation-Based Component Ranking

In a cloud application, there must be some components that frequently invoke a lot of other components. Their failures may affect a lot of subsequent components by error propagation and further harm the system reliability. So these components are considered to be important and they accord with the second description of significant components in Section 2.1. Intuitively, these significant components in a structure graph are the ones which have many outgoing links to other important components. Illuminated by the TrustRank algorithm [16], we propose another algorithm to calculate the second series of significance values of the cloud components, named as propagation-based significance values.

Assuming that a cloud application contains n components, the propagation-based significance value $VP(C_i)$ of a component C_i is defined as:

$$VP(C_i) = \alpha \sum_{C_i \to C_j} \frac{VP(C_j)}{\deg^{+}(C_j)} + (1-\alpha)\frac{1}{n} \tag{7}$$

Where $\frac{1}{n}$ is the basic significance value of C_i itself while $\sum_{C_i \to C_j} \frac{VP(C_j)}{\deg^{+}(C_j)}$ is the significance value derived from other components which are invoked by C_i. Similarly, the parameter α ($0 \le \alpha \le 1$) in (7) is employed to adjust the proportion of the two values, which is usually set as 0.85. By (7), a component C_i has a large propagation-based significance value if the sum of propagation-based significance values of the components which are invoked by C_i is large, showing that C_i invokes a large quantity of other significant components.

The equivalent matrix equation of (7) is:

$$\begin{bmatrix} VP(C_1) \\ \vdots \\ VP(C_n) \end{bmatrix} = \alpha \tilde{W} \begin{bmatrix} VP(C_1) \\ \vdots \\ VP(C_n) \end{bmatrix} + (1-\alpha) \begin{bmatrix} 1/n \\ \vdots \\ 1/n \end{bmatrix} \tag{8}$$

Where the matrix $\tilde{W} = (\tilde{w}_{ij})_{n \times n}$ is defined by:

$$\tilde{w}_{ij} = \begin{cases} \dfrac{1}{\deg^{+}(C_j)} & if\ C_i \to C_j \\ 0 & else \end{cases} \tag{9}$$

The procedures of calculating the propagation-based significance values are identical with those of calculating the failure-based significance values. First, randomly assign initial

values between 0 and 1 to the propagation-based significance values $VP(C_i)$ $(i = 1, 2 \cdots n)$. Then, solve (8) by repeating the computation until all significance values become stable.

With the above approach, the propagation-based significance values of the cloud components can be obtained. On the basis of the values, the components are ranked. A component is considered to be more significant if it has a larger value. The failures of these significant components selected from this ranking result will severely affect other components in the cloud application and further affect the system reliability.

3.4. Significant Component Determination

Based on the two series of significance values, the components in the cloud application can be ranked respectively. The failure-based significance values enable us to identify the significant components which have great direct impact on the system reliability while the propagation-based significance values help us locate the significant components which severely affect other components and further harm the system reliability. Which ranking result is more important? We believed that there is no accurate answer.

To reduce both of the direct and indirect threats and better improve the system reliability, Top-$\frac{k}{2}$ ($2 \le k \le n$ and k is even) components are respectively selected from the two ranking results and hence a total of k components are determined as the most significant components. In this way, the designer of the cloud application can improve the system reliability efficiently by enhancing the reliability of these components.

4. Experiments and Evaluation

In this section, extensive experiments are provided to validate DCR, evaluate the impact of different parameter settings on DCR and compare DCR with FTCloud.

4.1. Experimental Setup

In this section we compare the following approaches:

1. DCR: The components are ranked by DCR and the Top-K percent components are selected as the significant components for enhancing the reliability.
2. RandCR: K percent components are randomly selected as the significant components for enhancing the reliability.
3. FTCloud: The components are ranked by FTCloud and the Top-K percent components are selected as the significant components for enhancing the reliability.

The system reliability is considered to be the probability of generating correct output with correct input [17]. For a fair comparison, we assume that the internal failure probability ($intf$) of the selected components can be reduced to 80% after enhancing the reliability no matter which approach is employed. In addition, in DCR and FTCloud, the parameter α is used to balance the significance values derived from other components and the basic values of the components themselves. In previous studies [18, 19], it has been proved that 0.85 is a good choice. Thus, in our experiments, α is also set to be 0.85.

A scale-free graph is a graph whose degree distribution follows a power law, at least asymptotically. Previous studies have demonstrated that not only the Internet [20] but also the internal structures of common software such as Linux Kernel, Mozilla, Xfree86 and MySQL [21, 22] appear to be scale-free. Therefore, the network analysis software Pajek [23] is utilized to generate scale-free directed graphs as structure graphs of cloud applications in the experiments.

Three scale-free directed graphs with different settings of node numbers (i.e. 500, 1000 and 2000) are generated by Pajek in our experiments. Then the component invocation frequencies of each graph are randomly generated to simulate the statistical data during a period of running online. These component invocation frequencies are used in FTCloud and calculating the system reliability.

4.2. Validation and Performance Comparison

In order to validate DCR and compare DCR with FTCloud, the approaches are applied to the three graphs respectively and the experimental results of application reliability are

reported in Table 1. $intf$ represents the internal failure probability of the cloud components, including two value settings (i.e. 0.05 and 0.1). Top-K (K=2%, 6%, 10% and 20%) indicates that K percent most significant components in DCR and FTCloud, and K percent randomly selected components in RandCR are selected for enhancing the reliability. ep which is not listed in the table represents the error propagation probability of the cloud components. It is usually close to 1 [24] and thus we set it to be 0.99.

Compared with RandCR, DCR obtains better reliability performance in all the experimental settings. This observation shows that enhancing the reliability of the significant components identified by DCR can obtain better system reliability than enhancing the reliability of randomly selected components. In other words, DCR is able to effectively identify significant components which have great impact on the system reliability.

Table 1. Performance Comparison of Application Reliability

Node Numbers	Methods	Component intf=0.05				Component intf=0.1			
		Top2%	Top6%	Top10%	Top20%	Top2%	Top6%	Top10%	Top20%
500	RandCR	0.7381	0.7403	0.7410	0.7418	0.5416	0.5481	0.5499	0.5512
	FTCloud	0.7453	0.7597	0.7641	0.7714	**0.5572**	0.5750	0.5818	0.5930
	DCR	**0.7478**	**0.7681**	**0.7725**	**0.7818**	0.5568	**0.5883**	**0.5952**	**0.6096**
1000	RandCR	0.7247	0.7298	0.7312	0.7391	0.5201	0.5263	0.5234	0.5365
	FTCloud	0.7322	0.7460	0.7510	0.7590	**0.5398**	0.5552	0.5626	0.5746
	DCR	**0.7354**	**0.7522**	**0.7581**	**0.7680**	0.5396	**0.5650**	**0.5739**	**0.5889**
2000	RandCR	0.6974	0.7015	0.7098	0.7156	0.4881	0.4901	0.4928	0.5006
	FTCloud	0.7147	0.7251	0.7307	0.7383	0.5098	0.5248	0.5329	0.5440
	DCR	**0.7206**	**0.7322**	**0.7390**	**0.7494**	**0.5185**	**0.5355**	**0.5455**	**0.5610**

Compared with FTCloud, DCR provides better reliability performance in all the settings except for the case that Top-K equals to 2% while $intf$ is set as 0.1. In this case, the performance of DCR may be slightly worse than that of FTCloud when the node number is 500 or 1000 and the difference is not more than 0.0004. In order to further study the performance of DCR and draw performance comparison, more investigations into the impact of internal failure probability, error propagation probability and Top-K will be followed.

4.3. Impact of Relevant Parameters

To study the impact of the component internal failure probability ($intf$) on the system reliability, we compare the approaches in $intf$ settings of 0.01 to 1 with a step value of 0.01. In this experiment, the node number is 1000 and ep is set as 0.99. The experimental results of cloud application reliability in Figure 3 show that:

1· DCR consistently provides better reliability performance than FTCloud in all cases when Top-K=20%, Top-K=10% or Top-K=6%, and in almost all cases when Top-K=2%.
2· Only if Top-K=2% and $intf$ is not less than 0.09, the reliability performance of FTCloud approaches or slightly exceeds that of DCR. It exceeds a maximum of 0.0004.

To study the impact of the component error propagation probability (ep) on the system reliability, we compare the approaches in ep settings of 0.9 to 0.99 with a step value of 0.01. In this experiment, the node number is also 1000 and $intf$ is set as 0.1. The experimental results of cloud application reliability in Figure 4 show that:

1· DCR consistently provides better reliability performance than FTCloud in all cases when Top-K=20%, Top-K=10% or Top-K=6%, and in almost all cases when Top-K=2%.
2· Only if Top-K=2% and $ep = 0.9$, the reliability provided by FTCloud is 0.001 more than that provided by DCR.

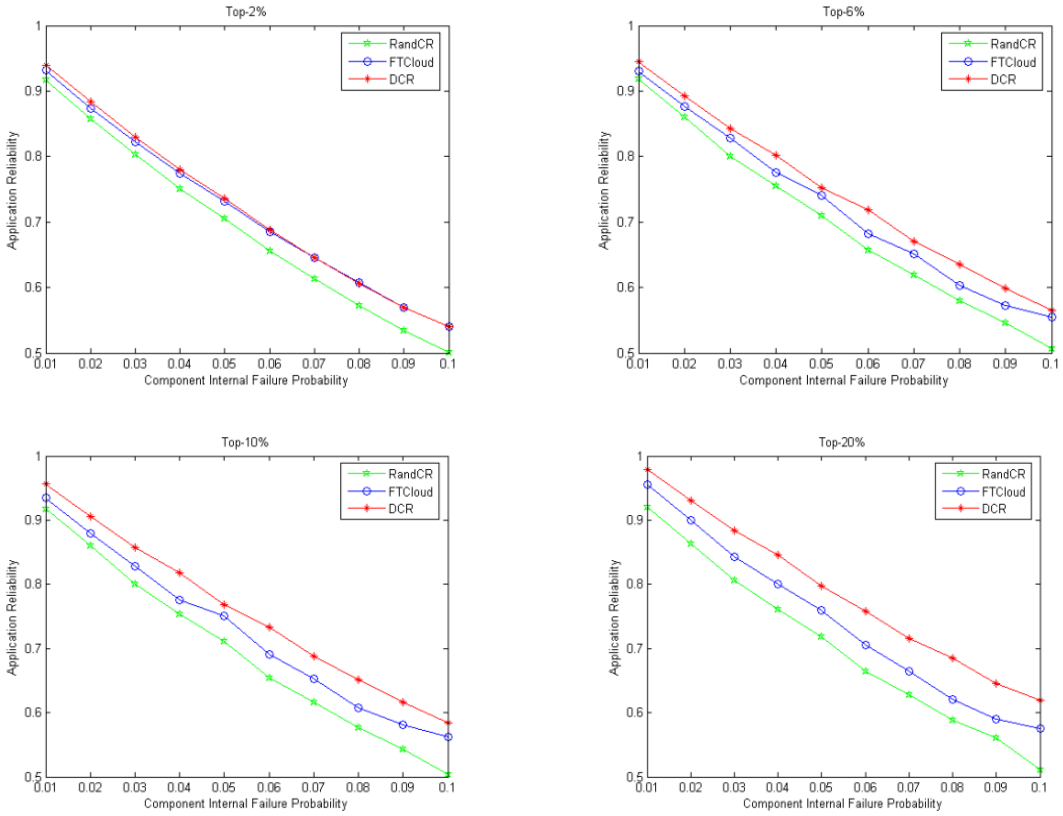

Figure 3. Impact of component internal failure probability

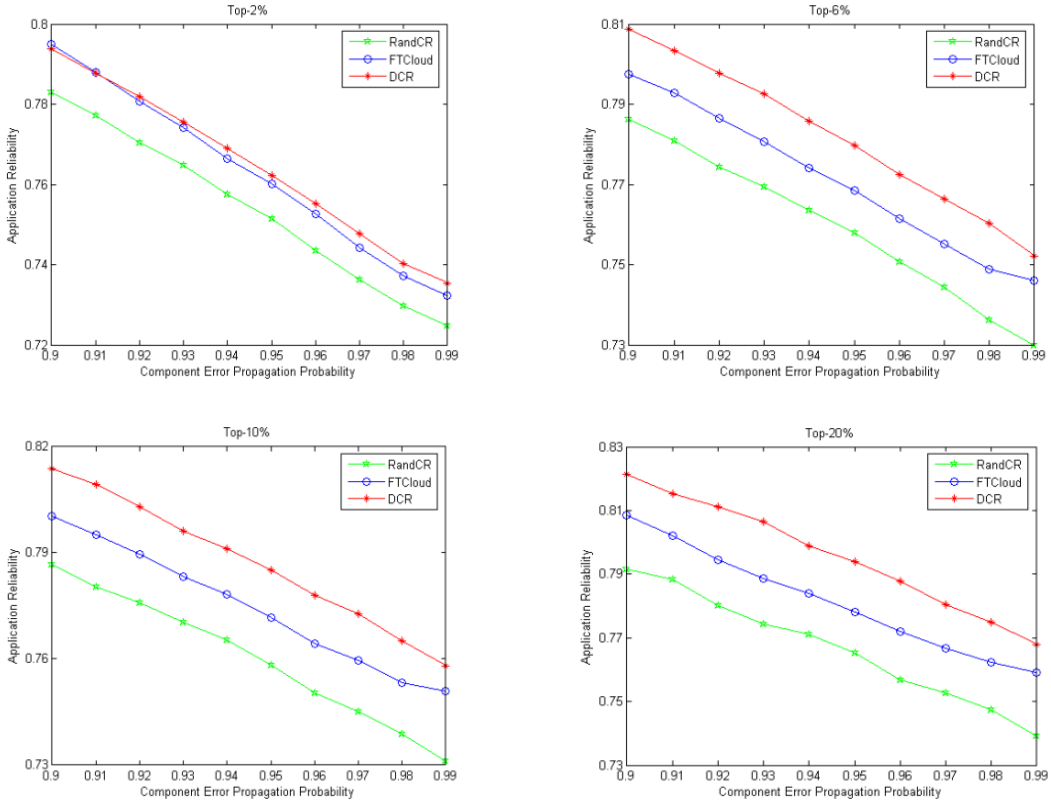

Figure 4. Impact of component error propagation probability

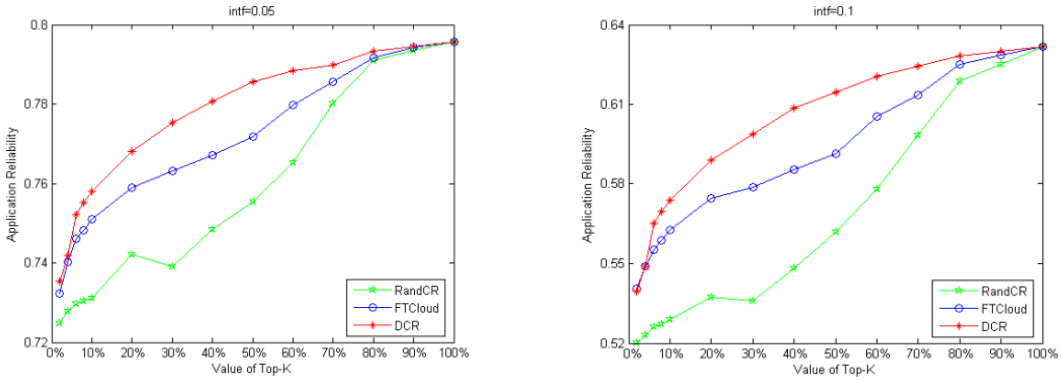

Figure 5. Impact of Top-K

To study the impact of Top-K on the system reliability, we compare the approaches in different Top-K settings. In this experiment, the node number is still 1000 and ep is set as 0.99, too. The experimental results of cloud application reliability in Figure 5 show that:

1. DCR consistently provides better reliability performance than FTCloud in all cases when $intf = 0.05$ and almost all cases when $intf = 0.1$.
2. Only if $intf = 0.1$ and Top-K is set as 2%, the reliability provide by FTCloud is 0.0002 more than that provide by DCR.

To sum up, DCR outperforms FTCloud in almost all cases. Only if Top-K=2% as well as ep is small or $intf$ is large, the performance of FTCloud may approach or slightly exceed that of DCR. This observation is due to the significant components determination of DCR and the inequality between impact of component internal failures on the reliability and impact of error propagation in cloud applications. DCR treats impact of component internal failures and error propagation equally, and selects Top-$\frac{k}{2}$ components from the two ranking results respectively. In these extreme cases, the impact of the first Top-$\frac{k}{2}$ components of the propagation-based ranking result on the reliability may be a little weaker than the impact of the second Top-$\frac{k}{2}$ components of the failure-based ranking result, causing the performance of DCR to be slightly worse than that of FTCloud in this case. The observation can only be found when node number is not big. When the scale of the cloud application reaches 2000 nodes, DCR outperforms FTCloud without any exception. Anyway, the negligible performance difference in a few extreme cases does not cover the effectiveness and advantages of DCR.

5. Conclusion

This paper proposes a component ranking framework for identifying significant components which have great impact on the cloud application reliability to help designers build reliable cloud applications. This framework takes into account the impact of component internal failures as well as the harm of error propagation to rank the components twice only employing the system structure information. The significant components are determined based on the two ranking results. The reliability of cloud applications can be greatly improved by enhancing the reliability of these significant components. Compared with FTCloud, the proposed framework considers more but requires less. Plenty of experiments are conducted to draw performance comparison and the results show that our framework is effective and outperforms FTCloud in almost all cases.

The future work includes: a) improving the determination of significant components, b) more experimental analysis of actual cloud applications, and c) considering more factors to identify significant components.

Acknowledgements

The work described in this paper was supported by the National Natural Science Foundation of China (No. 61173020) and Chinese National Programs for High Technology Research and Development (No. 2013AA01A215).

References

[1] Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. A View of Cloud Computing. *Communications of the ACM*. 2010; 53(4): 50-58.
[2] Prasad MP, Naik PR, Bapuji V. Cloud Computing: Research Issues and Implications. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*. 2013; 2(2): 134-140.
[3] Zheng Z, Zhang Y, Lyu MR. *CloudRank: A QoS-Driven Component Ranking Framework for Cloud Computing*. 2010 29th IEEE Symposium on Reliable Distributed Systems. New Delhi. 2010: 184-193.
[4] Rooney P. Microsoft's CEO: 80-20 Rule Applies to Bugs, Not Just Features. *ChannelWeb*. 2002.
[5] Zheng Z, Zhou TC, Lyu MR, King I. *FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications*. 2010 IEEE 21st International Symposium on Software Reliability Engineering. San Jose. 2010: 398-407.
[6] Avizienis A, Laprie JC, Randell B, Landwehr C. Basic Concepts and Txonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*. 2004; 1(1): 11-33.
[7] Lyu MR. Handbook of Software Reliability Engineering. New York: McGraw-Hill. 1996.
[8] Randell B, Jie X. The Evolution of the Recovery Block Concept. In: Lyu MR. *Editor*. Software Fault Tolerance. New York: Wiley. 1995: 1-21.
[9] Avizienis A. The Methodology of N-Version Programming. In: Lyu MR. *Editor*. Software Fault Tolerance. New York: Wiley. 1995: 23-46.
[10] Zeng L, Benatallah B, Ngu AHH, Kalagnanam J, Chang H. QoS-aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*. 2004; 30(5): 311-27.
[11] Maximilien EM, Singh MP. Conceptual Model of Web Service Reputation. *SIGMOD Record*. 2002; 31(4): 36-41.
[12] Naji-Hasan AH, Shu G. OWLS-CSM: A Service Profile Based Similarity Framework for Web Service Discovery. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2014; 12(4): 1079-1087.
[13] Lu G, Hai Y, Sun Y. A Reliable Web Services Selection Method for Concurrent Requests. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2014; 12(4): 1053-1063.
[14] Cheung RC. A User-Oriented Software Reliability Model. *IEEE Transactions on Software Engineering*. 1980; SE-6(2): 118-125.
[15] Page L, Brin S, Motwani R, Winograd T. *The PageRank Citation Ranking: Bringing Order to the Web*. Standford InfoLab. Report number: 1999-66. 1999.
[16] Gyongyi Z, Garcia-Molina H, Pedersen J. *Combating Web Spam with Trustrank*. Proceedings of the Thirtieth International Conference on Very Large Databases (VLDB). Toronto. 2004; 30: 576-587.
[17] Cortellessa V, Grassi V. A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-Based Systems. In: Schmidt HW, Crnkovic I, Heineman GT, Stafford JA. *Editors*. Component-Based Software Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg. 2007: 140-156.
[18] Brin S, Page L. Reprint of: The anatomy of a Large-scale Hypertextual Web Search Engine. *Computer Networks*. 2012; 56(18): 3825-3833.
[19] Inoue K, Yokomori R, Yamamoto T, Matsushita M, Kusumoto S. Ranking Significance of Software Components Based on Use Relations. *IEEE Transactions on Software Engineering*. 2005; 31(3): 213-25.
[20] Faloutsos M, Faloutsos P, Faloutsos C. On Power-Law Relationships of the Internet Topology. *SIGCOMM Computer Communication Review*. 1999; 29(4): 251-262.
[21] D Hyland-Wood, D Carrington, Y Kaplan. *Scale-Free Nature of Java Software Package, Class and Method Collaboration Graphs*. The 5th International Symposium on Empirical Software Engineering. Rio de Janeiro. 2005: 439-446.
[22] Myers CR. Software Systems as Complex Networks: Structure, Function and Evolvability of Software Collaboration Graphs. *Physical Review E*. 2003; 68(4): 46-116.
[23] De Nooy W, Mrvar A, Batagelj V. Exploratory Social Network Analysis with Pajek. Revised and Expanded Second Edition. New York: Cambridge University Press. 2011.
[24] Pham TT, Xefago D. *Reliability Prediction for Component-Based Systems: Incorporating Error Propagation Analysis and Different Execution Models*. 2012 12th International Conference on Quality Software (QSIC). Xi'an. 2012: 106-115.