LETTER

A Novel Component Ranking Method for Improving Software Reliability*

Lixing XUE[†], ***Member***, **Decheng ZUO**[†], **Zhan ZHANG**[†a)], ***and*** **Na WU**[†], ***Nonmembers***

SUMMARY This paper proposes a component ranking method to identify important components which have great impact on the system reliability. This method, which is opposite to an existing method, believes components which frequently invoke other components have more impact than others and employs component invocation structures and invocation frequencies for making important component ranking. It can strongly support for improving the reliability of software systems, especially large-scale systems. Extensive experiments are provided to validate this method and draw performance comparison.

key words: *large-scale system, component ranking, reliability, inverse pagerank*

1. Introduction

With the continual increase of hardware performance, the power of software becomes stronger and stronger, making software systems large scale and very complex. This includes not only software systems running on local computers, but also cloud applications and embedded software. Unfortunately, massive components and intricate invocation relationships bring a serious threat to the system reliability. Nowadays, the demand for highly reliable systems is becoming indescribably big. How to efficiently improve the reliability of large-scale systems has become a critical and challenging research problem.

The first step toward improving the reliability of a software system is to find out the important components which have great impact on the system reliability. Traditionally, this can be achieved by building a Markov model and then performing sensibility analysis [1]. But the state-space-explosion problem hinders applying this method to a large-scale system. Several approaches from other perspectives have been attempted. FTCloud [2], which is one of them, employs a component ranking algorithm derived from PageRank to identify important components and gains desired results. However, this method focuses on only components that are frequently invoked by other components, but ignores components that frequently invoke other components.

Manuscript received February 26, 2017.
Manuscript revised June 26, 2017.
Manuscript publicized July 24, 2017.

[†]The authors are with School of Computer Science and Technology, Harbin Institute of Technology, China.

*The work was supported in part by Technology Fund of State Key Laboratory of High-end Server & Storage (No. 2014HSSA05).

a) E-mail: zz@ftcl.hit.edu.cn (Corresponding author)

DOI: 10.1587/transinf.2017EDL8043

In contrast to FTCloud, we focus on components that often invoke other components and propose a novel component ranking method in this paper. This method performs component ranking based on component invocation relationships and invocation frequencies. After ranking, each component is given a significance score and the ones with high scores are considered important. We also provide extensive experiments to evaluate the impact of the identified components and draw performance comparison with FTCloud. The experimental results show that the proposed method can effectively and efficiently identify important components and it can gain better performance when the system has more components which intensively invoke others.

2. Method

The target of the component ranking method is to evaluate the significance of components in a software system and provide comparable results based on software structure and component invocation frequencies. This method is proposed according to the intuition that components which frequently invoke many other components are also important. The method can be divided into two main steps. Then, fault tolerance strategies are applied to the significant components.

2.1 Component Bayesian Graph Building

The structure of a software system, i.e. the component invocation relationships, can be modeled as a directed graph G, where a node c_i in the graph signifies a component in the software, and a directed edge e_{ij} from node c_i to node c_j denotes a component invocation relationship, i.e. component c_i invokes component c_j. Usually, this graph is named call graph or call multigraph. In some works, such as [3], it is also called component-based control flow graph.

Based on this graph, we assign a nonnegative weight value $W(e_{ij})$, which is no more than 1, to each edge e_{ij}. We are able to compute the weight value of edge e_{ij} by

$$W(e_{ij}) = \frac{freq(c_i, c_j)}{\sum_{k=1}^{n} freq(c_k, c_j)} \quad (1)$$

where $freq(c_i, c_j)$ is the frequency of component c_i invoking component c_j and n is the number of components in the software. If c_i does not invoke c_j, $freq(c_i, c_j) = 0$. For each

component c_j which is invoked by other components, the weights on its incoming edges satisfy $\sum_i W(e_{ij}) = 1$. As a matter of fact, the weight value $W(e_{ij})$ can be regarded as the posterior probability of c_i invoking c_j given that component c_j is executed. Meanwhile the calculation is similar to Bayes' formula, but using frequency statistics. Because of this, we name this graph "component Bayesian graph".

For a component Bayesian graph containing n components, an $n \times n$ matrix W is able to be constructed by employing Eq. (1). Each entry w_{ij} in the matrix is the weight value of edge e_{ij}, i.e. the value of $W(e_{ij})$. If there is no edge from component c_i to component c_j, i.e. c_i does not invoke c_j, $w_{ij} = 0$.

2.2 Component Ranking

In a software system, there are some components which are often invoked by many other components. FTCloud considers this category of components to be important to the system reliability, since the failures of these components will have greater impact than others. In contrast to these components, there must also be some components which frequently invoke a large number of other components in the same software system and we believe that they are important to the system reliability too. Due to the fact that these components often invoke many other components, we know that they must be invoked frequently first. So their failures have more impact on the system reliability [1]. From another perspective, the components which frequently invoke others intensively provide information to other components. If they are unreliable, incorrect data will be certain to spread out and negatively impact the system.

From an intuitive perspective, the important components we focus on are the ones which have many edges outgoing to other important ones in the component Bayesian graph. Illuminated by inverse PageRank of TrustRank [4], we propose an algorithm to evaluate the significance of components in a software system.

We consider a software system which has n components and define a significance score $S(c_i) \in [0, 1]$ for each component c_i, $i = 1, 2, \cdots, n$, with $n \in \mathbf{N}$. The score $S(c_i)$ for a component c_i is able to be calculated by

$$S(c_i) = \alpha \sum_{c_j \in I(c_i)} S(c_j) W(e_{ij}) + (1 - \alpha)\frac{1}{n} \tag{2}$$

where $I(c_i)$ is the set of direct successors of node c_i in the component Bayesian graph, i.e. the set of components that are invoked by component c_i, and α is a given parameter which is in the range of $[0, 1]$. The significance score of component c_i is composed of two parts: one is the basic score of itself (i.e. $\frac{1}{n}$) and the other one is the scores derived from the components that are invoked by c_i (i.e. $\sum_{c_j \in I(c_i)} S(c_j) W(e_{ij})$), which means significance propagates along with the opposite directions of edges in the component Bayesian graph. The parameter α is employed to balance the two score values. Equation (2) indicates that if the significance score of a component c_i is higher, the values of $S(c_j)$, $W(e_{ij})$ and $|I(c_i)|$ should be large, implying that c_i invokes many other important components frequently.

The equivalent matrix equation of Eq. (2) is:

$$\begin{bmatrix} S(c_1) \\ S(c_2) \\ \vdots \\ S(c_n) \end{bmatrix} = \alpha W \begin{bmatrix} S(c_1) \\ S(c_2) \\ \vdots \\ S(c_n) \end{bmatrix} + \frac{(1-\alpha)}{n} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{3}$$

where W is the matrix of the weight values and has been defined in last subsection. In this way, we can solve Eq. (3) by computing the eigenvector with eigenvalue 1.

Through the above method, we are able to obtain the significance scores of the software components. A component with a higher score is considered to be more important. Therefore, the components can be ranked. A specified number (such as Top-K percent) of the most important components can be identified. By improving their reliability or adding fault tolerance mechanisms to them, the global reliability can be greatly improved.

2.3 Fault Tolerance Strategies

Software fault tolerance strategies usually utilize functionally equivalent components to tolerate or handle component failures and improve the system reliability, and they are widely applied to critical systems, especially significant components in the systems. Similarly to FTCloud, this paper also introduces three common fault tolerance strategies, which are described in the following with formulas for computing the failure probabilities of the fault-tolerant structures. For a component, failure probability which is in the range of $[0, 1]$ is defined as the probability that a failure will occur during an invocation.

- Parallel: Parallel strategy invokes n functional equivalent components simultaneously and takes the first returned result as the overall output. The failure probability f of a parallel structure can be achieved by:

$$f = \prod_{i=1}^{n} f_i \tag{4}$$

  where n is the number of redundant components and f_i is the failure probability of the ith component.
- N-Version Programming (NVP): NVP [6] (also known as multiversion programming) invokes n components which are functionally equivalent but programmed independently, and determines the final result according to the n responses by majority voting. The failure probability f of a NVP approach can be calculated by:

$$f = \sum_{i=\frac{n+1}{2}}^{n} F(i) \tag{5}$$

  where n is usually odd and $F(i)$ is the probability that i alternative components from all the n components fail.

Table 1 Experimental result of system failure probability.

Node Numbers	Methods	Component FP=0.01			Component FP=0.05			Component FP=0.1		
		Top-1%	Top-5%	Top-10%	Top-1%	Top-5%	Top-10%	Top-1%	Top-5%	Top-10%
100	RandID	0.1020	0.0983	0.0955	0.2139	0.2105	0.2083	0.3281	0.3257	0.3215
	NCRID	0.0926	0.0903	0.0857	0.2027	0.2002	0.1986	0.3110	0.3078	0.3047
500	RandID	0.1965	0.1946	0.1898	0.2629	0.2597	0.2590	0.4581	0.4519	0.4431
	NCRID	0.1859	0.1839	0.1793	0.2509	0.2486	0.2463	0.4405	0.4377	0.4343
1000	RandID	0.2121	0.2095	0.2027	0.2699	0.2702	0.2688	0.4788	0.4737	0.4766
	NCRID	0.2028	0.2007	0.1954	0.2670	0.2646	0.2625	0.4704	0.4608	0.4577

For example, when $n = 3$, we have $f = F(2) + F(3)$, where $F(3) = f_1 f_2 f_3$ and $F(2) = f_1 f_2 (1 - f_3) + f_1 (1 - f_2) f_3 + (1 - f_1) f_2 f_3$.

- Recovery Block (RB): RB [5] is a well-known strategy, in which standby components will be invoked sequentially if the primary one fails. The failure probability f of a RB can be computed by:

$$f = \prod_{i=1}^{n} f_i \tag{6}$$

where n is the number of redundant components and f_i is the failure probability of the ith component.

In certain software, once n and each f_i are determined according to the cost and other constraints, we can obtain the failure probability values of the three strategies and then select the optimal one to improve the global system reliability. An automatically algorithm for the selection can be found in [2].

3. Evaluation

In order to evaluate the performance of identifying important components which have great impact on the system reliability, in this section we compare the following three methods:

- NCRID (the proposed component ranking method): Top-K percent important components are identified by the method in this paper and then their failure probabilities are reduced to improve the global reliability.
- RandID: K percent components are randomly selected and then their failure probabilities are reduced.
- FTCloud: Top-K percent important components are identified by FTCloud [2] and then their failure probabilities are reduced.

That is to say we utilize each method to select a group of K percent components and compare the impact of these component groups on the system reliability.

In NCRID and FTCloud, the parameter α is used to balance the derived significance and the basic significance of the component itself. In similar works [2], [7], [8], 0.85 has been proved as an optimal value. Thus, we also set this parameter to be 0.85. Besides, to exclude the influence of the different fault tolerance strategies and ensure fair comparisons, we assume that the failure probability (FP) values of the selected components are reduced to a given percentage after building fault tolerance strategies. In this way, the comparisons are able to directly show us which method is more effective in identifying important components that have great impact on the global reliability. It is obvious that this percentage can be set as any value in the range of $[0, 1]$. According to the experimental results that we have obtained under different failure probability values and settings of node numbers, the difference between the original overall failure probability and the improved overall failure probability appears to be distinct when the value is set as 80% or less. In the experiments, we suppose that the failure probability of each selected component is reduced to 80% after applying any fault tolerance strategy to the component.

Due to the fact that the internal structures of a great deal of common software, such as Linux Kernel, Mozilla, Xfree86 and MySQL, can be abstracted as scale-free graphs [9], [10], we use Pajek [11] - a network analysis tool, to generate scale-free directed graphs as the structures of target software systems. We employ random walk to simulate component invocation behaviors in software systems. For each graph, we archive 10,000 invocation sequences and use them to calculate the invocation frequencies. According to the structures and frequencies, component Bayesian graphs can be constructed.

3.1 Effectiveness

To evaluate the impact of the identified important component on the system reliability, we generate three component Bayesian graphs with different settings of node numbers (i.e. 100, 500 and 1000) and compare NCRID and RandID on them respectively under different Top-K and component failure probability (FP) settings. There are three Top-K values, i.e. K = 1, 5 and 10, and three component FP settings, i.e. 0.01, 0.5 and 0.1. The experimental results of system failure probabilities are recorded in Table 1.

We can find that our method, NCRID, obtains better failure probability performance in all settings. This observation shows that the important components identified by NCRID have greater impact on the system reliability. In other words, our method is effective in identifying important components for improving the system reliability.

3.2 Impact of Top-K

To study the impact of the Top-K parameter on identification results, we compare NCRID and RandID under different Top-K values. We use the graph with 500 nodes and the results are depicted in Fig. 1.

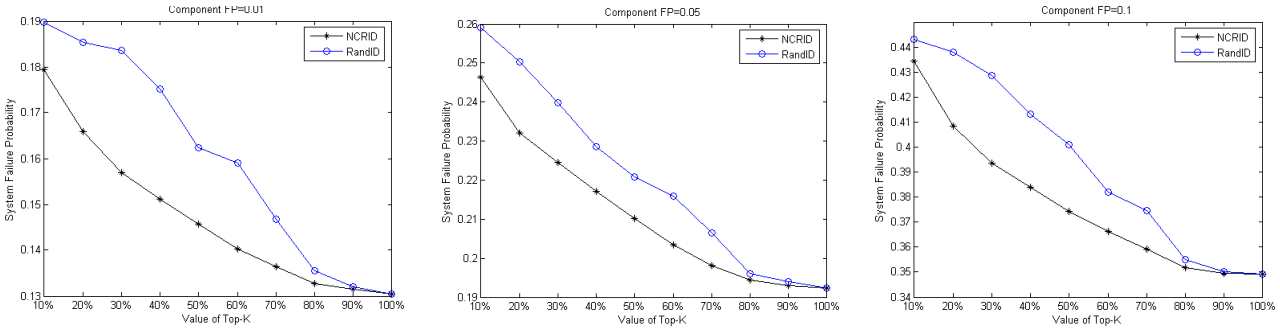


Fig. 1 Impact of Top-K.

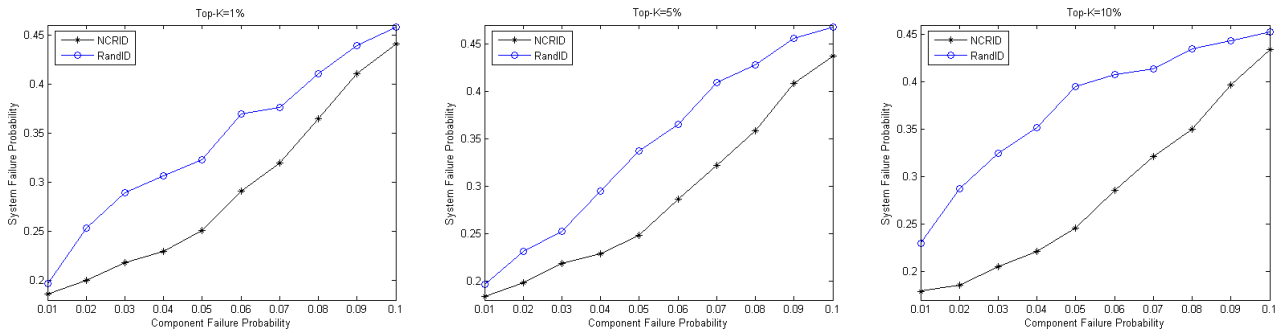


Fig. 2 Impact of component failure probability.

As the Top-K value grows from 10% to 90%, NCRID consistently provides better reliability performance than RandID under the different componentfailure probabilities. Only when the Top-K reaches 100%, the two failure probability values are identical. This demonstrates that the Top-K percent components identified by NCRID always have greater impact on the system reliability, namely the Top-K value do not affect the effectiveness of our method.

3.3 Impact of Component Failure Probability

To study the impact of the component failure probability parameter on identification results, we compare NCRID and RandID under failure probability settings of 0.01 to 0.1 with a step value of 0.01. The node number is 500, too. The results are showed in Fig. 2.

Under different Top-K values (i.e. Top-1%, Top-5% and Top-10%), NCRID consistently outperforms RandID regardless of the component failure probability settings. This phenomenon indicates that NCRID is free of the component failure probability parameter.

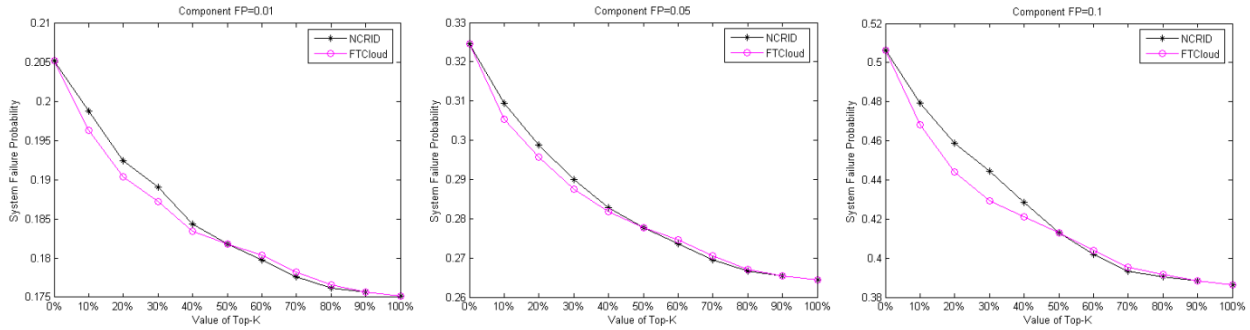
3.4 Performance Comparison

Our method has been proved to be effective regardless of component failure probability and Top-K values, so has FTCloud [2]. Now, we want to make clear that for a given software system, which one has better performance, namely the identified important components have greater impact on the system reliability.

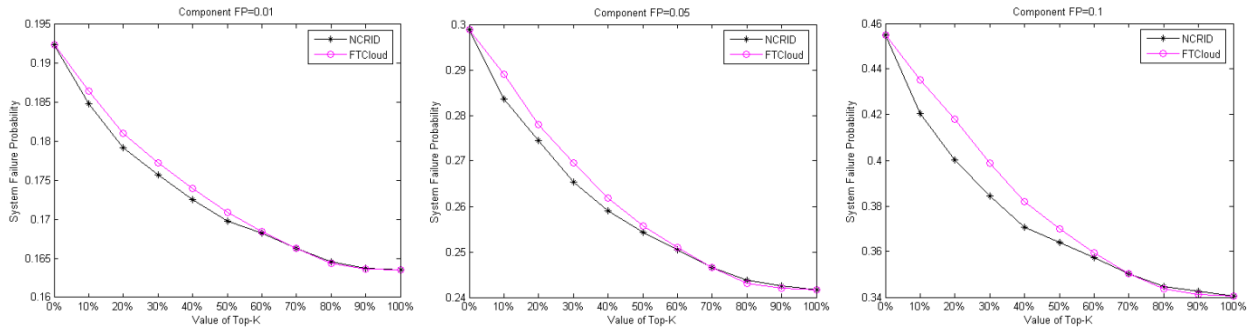
Our method pays attention to the outgoing links of components while FTCloud focuses on the incoming links. It is reasonable to speculate that the distribution of in-degrees and out-degrees in the component Bayesian graph may affect the performance of them. Therefore, we tune the parameters in Pajek to generate three 500-node directed graphs and name them “IN”, “OUT” and “UNI”, respectively. IN has a part of nodes whose in-degrees are high, OUT has a part of nodes whose out-degrees are high and the distribution of degrees in UNI is uniform. We compare NCRID and FTCloud on them under three different component failure probability (FP) values (i.e. 0.01, 0.05 and 0.1). The experimental results derived from system IN, OUT and UNI are drawn in Fig. 3(a), 3(b) and 3(c), respectively.

Overall, the performance of the two methods is not too different. Regardless of the component FP value, FTCloud outperforms NCRID in system IN, when Top-K value is smaller while the performance of them is similar when Top-K becomes big. Oppositely, in system OUT, NCRID outperforms FTCloud in most cases and only when Top-K is large, they approach to each other. In system UNI, the two methods are neck and neck through all Top-K values, no matter what component FP value is. In fact, only a small set of components are expected to be ameliorated when improving the system reliability. We should pay more attention to the smaller Top-K values.

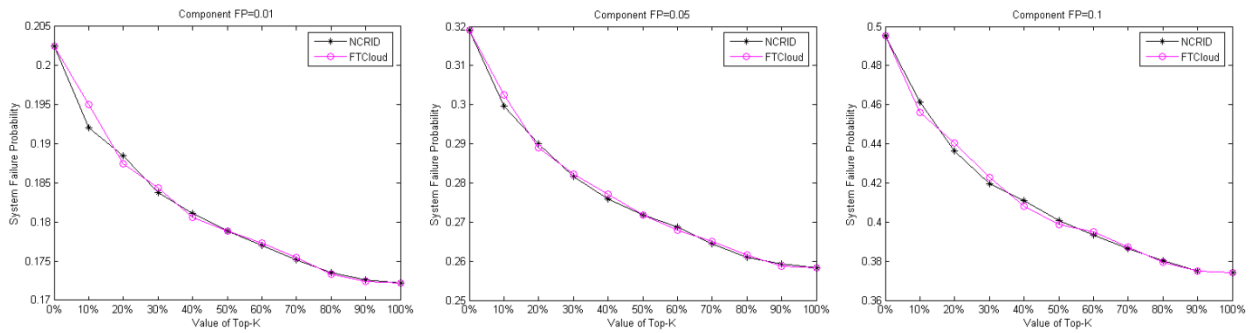
We can draw a tentative conclusion that our method outperforms FTCloud when the software system has more components which frequently invoke others. Unfortunately,



(a) System IN



(b) System OUT



(c) System UNI

Fig. 3 Performance comparison with FTCloud.

we cannot reveal the exact quantitative condition in a short term. This has been involved in the future work. Besides, how to combine the two methods into a stronger synthetic method may become a part of the future work.

4. Conclusion

This paper proposes a novel component ranking method to identify important components which have greater impact on the system reliability. In this method, the rank of a component is determined by the number of components that are invoked by this component, the significance of these components and how often they are invoked. Extensive experiments shows that the method is effective and the ranking results can provide strong support for improving the system reliability.

References

[1] R.C. Cheung, "A User-Oriented Software Reliability Model," IEEE Trans. Software Engineering, vol.SE-6, no.2, pp.118–125, 1980.

[2] Z. Zheng, T.C. Zhou, M.R. Lyu, and I. King, "FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications," Proc. 21st IEEE International Symposium on Software Reliability Engineering, pp.398–407, San Jose, 2010.

[3] A. Mohamed and M. Zulkernine, "A Control Flow Representation for Component-Based Software Reliability Analysis," Proc. 6th IEEE International Conference on Software Security and Reliability, pp.1–10, Gaithersburg, 2012.

[4] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating Web Spam with Trustrank," Proc. 30th International Conference on Very Large Databases (VLDB), pp.576–587, Toronto, 2004.

[5] B. Randell and J. Xu, "The Evolution of the Recovery Block Concept," ed. M.R. Lyu, Software Fault Tolerance, pp.1–21, Wiley, Chichester, 1995.

[6] A. Avizienis, "The Methodology of N-Version Programming," in Software Fault Tolerance, M.R. Lyu, pp.23–46, Wiley, Chichester,

1995.

[7] S. Brin and L. Page, "Reprint of: The anatomy of a Large-scale Hypertextual Web Search Engine," Comput. Netw., vol.56, no.18, pp.3825–3833, 2012.

[8] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, "Ranking Significance of Software Components Based on Use Relations," IEEE Trans. Software Engineering, vol.31, no.3, pp.213–215, 2005.

[9] A.P.S. de Moura, Y.-C. Lai, and A.E. Motter, "Signatures of Small-World and Scale-Free Properties in Large Computer Programs," Physical Review E, vol.68, no.1, ID.017102, 2003.

[10] C.R. Myers, "Software Systems as Complex Networks: Structure, Function and Evolvability of Software Collaboration Graphs," Physical Review E, vol.68, no.4, ID.046116, 2003.

[11] W. de Nooy, A. Mrvar, and V. Batagelj, Exploratory Social Network Analysis with Pajek, Revised and Expanded Second Edition, Cambridge University Press, New York, 2011.