FSCRank: A Failure-Sensitive Structure-Based Component Ranking Approach for Cloud Applications

Na WU¹, Student Member, Decheng ZUO¹, Zhan ZHANG¹, Peng ZHOU¹, and Yan ZHAO¹, Nonmembers

SUMMARY Cloud computing has attracted a growing number of enterprises to move their business to the cloud because of the associated operational and cost benefits. Improving availability is one of the major concerns of cloud application owners because modern applications generally comprise a large number of components and failures are common at scale. Fault tolerance enables an application to continue operating properly when failure occurs, but fault tolerance strategy is typically employed for the most important components because of financial concerns. Therefore, identifying important components has become a critical research issue. To address this problem, we propose a failure-sensitive structure-based component ranking approach (FSCRank), which integrates component failure impact and application structure information into component importance evaluation. An iterative ranking algorithm is developed according to the structural characteristics of cloud applications. The experimental results show that FSCRank outperforms the other two structure-based ranking algorithms for cloud applications. In addition, factors that affect application availability optimization are analyzed and summarized. The experimental results suggest that the availability of cloud applications can be greatly improved by implementing fault tolerance strategy for the important components identified by FSCRank.

key words: component ranking, failure impact, application structure, buffer node, availability improvement

1. Introduction

Cloud computing has become a highly demanded service model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort [1]. Cloud-based services integrate globally distributed resources into seamless computing platforms [2].

Cloud applications are software systems that operate in cloud environments, which can be regarded as combinations of standard web applications and conventional desktop applications. Cloud applications are easy to use because they are generally web-based and do not require installing, updating and managing business applications across every access device. For cloud application providers, however, various challenges exist in designing, building, operating, and maintaining applications. High availability is one of the major concerns.

Cloud applications are typically composed of a large number of service components, which are distributed in multiple cloud data centers, sometimes even span private and public clouds [3]. Unfortunately, component failure can be expected to occur more frequently as the complexity of applications increases, thus reducing the availability of applications. Consequently, fault tolerance becomes necessary because it enables an application to continue operating properly when component failure occurs. However, improving availability and employing fault tolerance involves a trade-off between the cost of failure and the cost of redundancy [4].

Due to cost, we generally cannot employ fault tolerance for all components at one time to improve application availability; priority should thus be given to components that are more important [5]. Assigning redundant resources to a set of selected critical components helps achieve an ideal trade-off between higher availability and lower costs. Therefore, identifying important components is the prerequisite for achieving the cost-effective availability optimization of cloud applications.

Component ranking has been widely studied in many research fields, such as social networks, information retrieval, software engineering, web services, cloud applications, and so on. Various approaches and algorithms for component ranking have been proposed. One of the most famous is PageRank, used by Google Search to rank web sites [6]. PageRank and improved methods based on this algorithm are applied to rank components in many cases, including cloud applications.

PageRank scores and ranks web pages (nodes in a graph) based on a random walk model. A surfer begins his random walk at a web page and proceeds from the current node to next one. Typically, he visits some nodes more often than others, and these are nodes with many links coming in from other frequently visited nodes. The idea behind PageRank is that pages are more important if they are visited more often. This kind of stochastic behavior is an example of the theory of Markov processes, and the existence and uniqueness of solutions are guaranteed when two conditions are met; that is, 1) the graph is strongly connected, and 2) there are no dead ends (leaf nodes in a graph) that have no links out. The existence of dead ends will cause the random walk has nowhere to go to, and the scores will be leaked out from the dead ends, which is known as the dead end problem in PageRank [7]. There are two typical approaches to dealing with dead ends, dead ends cutting and “taxation”. With the method of dead end cutting, we can drop the dead ends and theirs incoming links from the graph. Since doing...
so may create new dead ends, the cutting processes should be done recursively until there are no longer dead ends. By contrast, “taxation” allows the presence of dead ends, but their scores are allocated to all other nodes evenly.

However, the typical solutions for the dead end problem are not accurate and effective enough because of the structural differences between cloud applications and web page linkage. Compared with web page linkage structures, application structures have some unique features. First, if no redundant component exists in application, the structure graphs are typically relatively more weakly connected than others, i.e., the average connectivity degree of application structures is relatively lower. Second, applications generally contain multiple leaf nodes, whose proportion is relatively higher. In such a situation, more dead ends need to be dropped if using dead ends cutting method. Consequently, it may generate a very small size graph, or even cannot obtain a graph without leaf nodes. On the other hand, evenly allocating scores of the dead ends to all nodes becomes unfair because the availability properties of nodes are different in an application. Therefore, to develop a more suitable and accurate component ranking approach for cloud applications, component availability properties and application structure properties should be considered comprehensively in component importance calculation.

In this paper, we propose a novel component ranking approach for cloud applications, named the failure-sensitive structure-based component ranking approach (FSCRank). FSCRank takes the availability properties as component core factors and the invocation relations as environment factors in component importance evaluation. We develop an iterative ranking algorithm according to the structural characteristics of cloud applications. With the ranking list of FSCRank, cloud application providers can recognize the important components and design availability optimization strategies for cloud applications.

In summary, the contributions of this paper include the following:

- We characterize the structural differences between cloud applications and other information systems, and we investigate the influence of the unique features on component importance evaluation.
- FSCRank is proposed to rank components by integrating component availability properties and application structural properties together into calculating component importance.
- A buffer node is introduced into structure graphs, which prevents the dead end problem and enables FSCRank to be more effective and suitable for cloud applications.
- We conduct extensive experiments on component importance evaluation, and we discuss various key factors of component ranking.

The remaining parts of this paper is organized as follows. Section 2 provides a review of the related research literature. Section 3 describes the details of our proposed approach and algorithm. Section 4 provides the experimental setup and results and discusses various factors of component importance measurement. Finally, we conclude this paper in Sect. 5.

2. Related Work

Cloud computing, the long-held dream of computing as a utility, has been transforming the IT service paradigm, making software even more attractive as a service and reshaping the way IT hardware is designed and purchased[8]. Cloud computing allows companies to get their applications up and running quickly, by providing improved manageability, simplified requirements, and less maintenance. Although cloud providers typically offer guaranteed quality of service (QoS)[9] and remuneration for service-level agreement (SLA) violations[10], ensuring high quality of service is still one of the major concerns. Today’s applications often comprise a large number of service components; each of which has a certain probability of failure. Moreover, if a complex application is performed trillions of times per day, even a small probability of failure is critical[11]. Application developers should realize the inevitability of failures[12], thereby identify important components and design fault tolerance strategies for applications.

One of the most important problems in reliability theory is to evaluate the relative importance of various components within a system. Current component importance measurements can be broadly divided into two categories: reliability importance and structural importance[5].

Reliability importance measures offer an evaluation of the relative importance of individual components or groups of components constituting a system based on the reliability properties of components[13]. For instance, Bayesian reliability importance defines the importance of a component as the probability that the component failure causes system failure[14]. By contrast, the criticality reliability importance considers both sides of functioning and failure. It ranks components by evaluating component functions (fails) and is critical for system functioning (failure) given that the system functions (fails)[15].

Structural information measurement is another challenging issue in information science and computer science[16]. In fact, structure importance measures refer to the importance of the positions of the components in the system.

The earliest structure-based solutions to rank nodes were based on centrality-based metrics[17], such as degree centrality[18], closeness centrality[17] and betweenness centrality[19] derived in the research field of social networks[20]. PageRank[6] and HITS[21] were subsequently proposed. PageRank assumes that more important websites are likely to receive more links from other websites. PageRank determines the importance of the website by counting the number and quality of links to a page. HITS works by considering the notion of authority and “hub pages” in conjunction with the link structure. Both of these algorithms
treat all links equally when distributing rank scores.

Various improved ranking methods based on PageRank and HITS have been proposed. For instance, weighted PageRank [22] is an extension to the original PageRank. This approach introduces a weighted transition probability matrix in which each entry is determined by both inlinks and outlinks of the pages. The popularity of the pages is also considered in the rank score distributions. N-step PageRank [23] is another improved algorithm of PageRank, which constructs the transition probability matrix using the approach of “looking N-step ahead”. Moreover, each entry in the matrix is proportional to the page’s N-step neighbor count. All the ranking methods mentioned above are automatic unsupervised algorithms based solely on structural information.

It is reasonable and effective to rank webpages by simply considering structure information rather than the content. In this way, malicious cheating on contents can be prevented because PageRank is independent of any particular search query. In recent years, PageRank-like approaches have been applied to address the problem of component ranking for cloud applications [24]–[27]. Beyond directly applying a PageRank-like method or simply combining component availability properties and PageRank-like methods, there is a need to analyze the structural characteristics of cloud applications and quantify their influence on availability improvement.

To this end, we propose FSCRank to evaluate failure-sensitive component importance. This approach, a further extension of PageRank, is more effective and suitable for cloud applications.

3. Failure-Sensitive Structure-Based Component Ranking Approach

The overview of the proposed approach is illustrated in this section. We then analyze and quantify the factors of component importance, including the availability factors of components and structural factors of applications. Finally, we describe the FSCRank approach and algorithm.

3.1 Approach Overview

For availability-oriented component ranking, a component is regarded as “important” if it has a high failure probability and a significant impact on the availability of the application. Furthermore, a component is more “important” if it is invoked by more important nodes since it is more likely to propagate failures to other components. This description indicates that component importance depends on both component availability properties and application structure properties.

To comprehensively evaluate the component importance, we propose FSCRank. FSCRank consists of three processes: structure information extraction, component availability properties quantification, and component ranking. The framework of FSCRank is illustrated in Fig. 1.

First, the application structure is extracted from source code and documentation and then presented as a weighted directed graph. Nodes and edges in the graph represent application components and invocation relations among them, respectively. The weight of an edge is the invocation frequency of a pair of nodes. The data of invocation frequency is collected from log files.

The availability properties of components are then quantified, including the failure probability (FP) and the failure impact (FI). Since the failure influence of a component on application availability is the comprehensive performance of FP and FI, the overall failure influence is computed and normalized for each component.

The final process is component ranking, which includes two subprocesses: buffer node addition and component ranking. According to the structural characteristics of cloud applications, a new type of node, termed the buffer node, is introduced into FSCRank. The buffer node is added into the structure graph, which is connected with each node by a bidirectional link. The presence of the buffer node serves two purposes: 1) addressing the dead end problem, and 2) playing a part in the redistribution of the importance value. Then, the equation of component importance is created based on the revised structure information and quantified component availability properties. The results are obtained by an iterative algorithm and sorted into a ranking list.

3.2 Structure Information Extraction

Application structure refers to the high-level structure of a cloud application system, which comprises components, relations among them, and properties of both components and relations [28]. An individual component can be a software package, a web service, a web resource, or a module that encapsulates a set of related functions. The application structure shows the relationships among components, which is vital for component failure impact assessment and application availability evaluation.

A cloud application can be abstracted into a weighted directed graph $G$. A node $c_i$ in graph $G$ represents a component. A directed edge $e_{ij}$ from node $c_i$ to node $c_j$ denotes that component $c_i$ invokes $c_j$. Each node has an importance value $IV(c_i)$, failure probability $FP(c_i)$ and failure impact $FI(c_i)$. Each edge $e_{ij}$ has a weight value $w_{ij}$, which represents the frequency of node $c_j$ is invoked by node $c_i$. 
\[ \omega_{ij} = \frac{inv_{ij}}{\sum_{j=1}^{n} inv_{ij}} \]  
\[ \sum_{j=1}^{n} \omega_{ij} = 1 \]

where \( inv_{ij} \) is the total number of times that node \( c_i \) invokes \( c_j \). If node \( c_i \) does not invoke \( c_j \), \( \omega_{ij} = 0 \).

Mathematically, an application structure graph with \( n \) components can be converted to an \( n \times n \) transition probability matrix \( M \). Each entry is the value of \( \omega_{ij} \). For each nonleaf node, the transition probability matrix \( M \) satisfies the following relationship:

\[ \sum_{j=1}^{n} \omega_{ij} = 1 \]

The application structures have some unique features compared to communication networks, social networks, or webpage linkages. Components in an application without redundancy are typically connected more weakly, and the average degree of connectivity is lower. Additionally, there are a number of leaf nodes in the application structure, whose proportion is relatively high.

### 3.3 Component Availability Properties Quantification

The availability properties of components play a major role in availability-oriented component ranking. The quantification of availability properties is the premise of component importance evaluation.

Component failure probability \( FP(c_i) \), defined as the ratio of the total times that component \( c_i \) failed to the total times that \( c_i \) is invoked, can be calculated as

\[ FP(c_i) = \frac{f(c_i)}{\sum_{k=1}^{n} inv_{ki}} \]  

where \( f(c_i) \) is the total times that component \( c_i \) failed and the sum represents the total times that \( c_i \) is invoked. The data of \( f(c_i) \) and \( inv_{ki} \) can be collected from application trace logs.

Component failure impact analysis is used to systematically analyze component failures and identify the resultant effects on application availability. A straightforward way is to divide components into critical components and noncritical components, depending on whether a component failure will cause the application failure [24]. However, this dichotomy of component failure impact has two shortcomings: 1) it depends on manual identification, and 2) it is not accurate enough because it lacks a precise measure of component failure impact. Therefore, we adopt a statistical method to evaluate the component failure impact \( FI(c_i) \).

The calculation of \( FI(c_i) \) is as follows:

\[ FI(c_i) = \frac{f(appc_i)}{f(c_i)} \]

where \( f(appc_i) \) denotes the application failure times caused by the failure of component \( c_i \). Thus, \( FI(c_i) \) is the ratio of the application failure times caused by the failure of component \( c_i \) to the total times that component \( c_i \) failed.

For each individual component \( c_i \), how greatly it influences the application depends on both the failure probability \( FP(c_i) \) and failure impact \( FI(c_i) \). Meantime, the availability properties of each component may be different. Some components have high failure probability, while others do not. Failures of different components result in different consequences for the application. Therefore, we normalize the overall failure influence of all components, designated \( P(c_i) \), as follows:

\[ P(c_i) = \frac{FP(c_i)FI(c_i)}{\sum_{i=1}^{n} FP(c_i)FI(c_i)} \]

### 3.4 Component Ranking

The availability properties can be regarded as component factors in the component ranking, while invocation relations can be interpreted as environment factors. This framework composes the importance value flow system, which takes the FP and FI as the core factors affecting the component importance and propagates the component importance value along the linkage of applications. On the one hand, the component importance depends on its overall failure influence on application availability. On the other hand, the component importance interacts through invocation relations. Therefore, the component importance constitutes a comprehensive measure of the component failures and the application structure.

In the importance value transfer process, the importance value of a component consists of two parts; one part is reserved for itself, and the other part is derived from the components that invokes it. A parameter \( d \) is used to determine the ratio of these two parts. In this way, we can compute the importance value \( IV(c_i) \) for a component as follows:

\[ IV(c_i) = dIV(c_i) + (1 - d) \sum_{k=1}^{n} IV(c_k) \omega_{ki} \]

In structure-based component ranking, leaf nodes without outlinks cannot ensure that the node score calculating equation converges; thus, stable node scores cannot be obtained, which is called the dead end problem in PageRank. Several solutions have been developed to address this problem, such as iterative dead end cutting and teleporting. So far, we can evaluate component importance through Eq. (6), but this approach is not suitable or accurate enough for cloud applications because the proportion of leaf nodes in application structures may be generally higher than that in information retrieval or social networks.

To resolve this issue more effectively, a buffer node is added into the structure graph, which is connected with each node by a bidirectional link. First, the existence of buffer node makes the structure graph strongly connected; thus, there is no longer a leaf node. Therefore, the dead end problem is avoided. Moreover, the buffer node can cache and redistribute part of the importance value for all nodes as well. In this way, the failure influence and the structure information are combined into the calculation of importance value of components. The details of buffer node addition and component ranking are as follows.
**Buffer Node Addition**: Given a cloud application consisting of \( n \) components and \( m \) directed invocation links, the nodes are first numbered starting from the application entry node in some way, for example, by the order of breadth-first traversal. Then, a buffer node connected with every single node by a bidirectional link is added. Thus, the application structure becomes strongly connected, consisting of \( n + 1 \) nodes and \( m + 2n \) edges.

An example of the structure graph with a buffer node is illustrated in Fig. 2. For simplicity, we only depict the bidirectional links between the buffer node and a pair of nodes (node 2 and 4) rather than all bidirectional links between the buffer node and each node, on the condition that the function of the buffer node can be clarified. In Fig. 2, the enlarged view of the partial application structure with a buffer node shows how the importance value flows among the buffer node \( c_{buffer} \) and the pair of parent-child nodes \( c_i \) and \( c_j \).

Since the converged importance values are independent of the initial values of the nodes, a random initial importance value between 0 and 1 is assigned to each node. The initial state of the buffer node is empty. Each node propagates (and retrieves) importance values to (and from) the buffer node and its child (parent) nodes. The importance value of node \( c_i \) is assigned to its child nodes and the buffer node, and a distribution coefficient \( d \) as an index of sensitivity to component failure is used to decide the ratio of importance value to its child nodes and the buffer node. In this partial relationship, node \( c_i \) apportions \( dIV(c_i)w_{ij} \) importance value to its child node \( c_j \) and an importance value of \((1 - d)IV(c_i)\) to the buffer node \( c_{buffer} \). Node \( c_j \) obtains \( dIV(c_i)w_{ij} \) from its parent node \( c_i \) and an importance value of \( P(c_j)IV(c_{buffer}) \) from the buffer node.

Therefore, from the global perspective, the importance value of the buffer node—and thus the importance value cached in the buffer node—is computed as follows:

\[
IV(c_{buffer}) = (1 - d) \sum_{i=1}^{n} IV(c_i)
\]

where the value of \( d \) determines the sensitivity distribution between structure information and failure impact. \( d > (1 - d) \) implies that the importance of nodes is more dependent on structure information. In contrast, \((1 - d) > d \) implies that the failure impact plays a major role in the importance value calculation.

Note that a difference exists in setting the value of \( d \) between leaf nodes and normal nodes. Because the edge from a leaf node to the buffer node is the only outlink of this leaf node, the buffer node needs to receive the entire corresponding importance value to avoid losing the importance value. Therefore, the value of \( d \) is set as 0 for leaf nodes.

**Component Ranking**: In a structure graph with the buffer node, the failure-sensitive importance value \( IV(c_i) \) for a component \( c_i \) is defined as follows:

\[
IV(c_i) = d \sum_{k=1}^{n} IV(c_k)w_{ki} + P(c_i)IV(c_{buffer})
\]

The left term of summation represents the importance value obtained from its parent nodes, reflecting the influence of the application structure on the component importance value. The sensitivity of the evaluation to component failure is reflected by the value retrieved from the buffer node, i.e., the right term of summation.

The importance values can be calculated iteratively or algebraically. We adopt the iterative method in this paper. The equation can be computed by starting with any set of ranks and iterating the computation until it converges. In each iteration, the value of each node propagates to its child nodes and buffer node through directed edges in proportion to the weights of edges. The calculation is repeated until the importance value of all nodes become stable. Then, each node retrieves its counterpart of importance value \( P(c_i)IV(c_{buffer}) \) from the buffer node, and the state of the buffer node returns to empty. Finally, a ranking list is formed according to the component importance values. An iterative algorithm is designed according to FSCRank and the process is described as Algorithm 1.

In summary, FSCRank includes the following steps:

1. Extract structure information from an application, and build the structure graph.
2. Normalize the overall failure impact of each component based on FP and FI (lines 1–3).
3. Add the buffer node into the structure graph, which is connected to every node by a bidirectional link.
4. Assign 0 to the buffer node, and random numerical values between 0 and 1 to normal nodes as their initial value (lines 4–7).
5. Compute the importance value for every node iteratively. The calculation is repeated until all importance values become stable (lines 8–11).
6. Return a ranking list according to the final component importance values (line 12).

With the presence of the buffer node, the modified structure graph becomes a strongly connected graph without leaf nodes, which meets the conditions of the existence.
and uniqueness of solutions of Markov processes. In fact, the buffer node is an extension of the “taxation” method of PageRank, which can be regarded as a cache in distributing the importance values of leaf nodes. Instead of directly allocating the importance values of leaf nodes to all other nodes evenly, FSCRank allows the buffer node to take the entire importance value of each leaf node, and then aggregate them with the part of the importance values of the non-leaf nodes. After that, each node retrieves its corresponding share of importance value from the buffer node, which is in proportion to its overall failure influence.

Therefore, from the perspective of the impact on graph structure, the addition of the buffer node enables the graph to satisfice the requirements of component ranking. In addition, from the perspective of the impact on importance value flow, the introduction of the buffer node makes “taxation” process sensitive to component failure influence. FSCRank is thus more suitable and accurate for component ranking for cloud applications.

3.5 The Applicability of FSCRank

As a new component ranking approach for cloud applications, FSCRank takes the source codes and documents of an application as its input, and provides a component ranking list as its output. The purpose of component ranking is to assist cloud application providers in optimizing the availability of applications. Component ranking provides decision support for cloud applications providers in the form of a ranking list. Cloud application providers typically wish to maximize the utility of investment on availability optimization; thus, a set of critical components with higher ranks can be selected from the ranking list. Allocating the selected important components with fault tolerant strategies can reduce their failure rates, which is an effective way to achieve a desired level of availability, especially when a cost constraint exists.

FSCRank can be applied to both of applications with or without redundancy. For applications without redundancy, each component is modeled as a node of structure graph, and the availability properties of a component are computed as the parameters of its corresponding node. For applications already have redundant components, a set of one primary component and its redundant components (one or more) can be modeled as a node of structure graph, and the availability parameters are calculated from the comprehensive performance of this set of components.

4. Experiments

We conducted extensive experiments to evaluate the performance of the proposed approach. The overall goal is to prove the effectiveness of FSCRank and to investigate the influence of major factors on application availability optimization, including the fault tolerant ratio (FTR), the component availability properties, and the leaf node ratio (LNR).

First, the experimental setup is outlined in this section. Then, we compare the performance of FSCRank and two representative structure-based component ranking approaches. Finally, we report findings and analysis about the influence factors of application availability optimization.

4.1 Experimental Setup

Cloud applications, as a representative networked software, have been proven to share the same global characteristics of complex networks, such as the “small world” and “scale free” characteristics [29], [30]. Thus, we use NetworkX, a Python library for studying graphs and networks [31], to generate scale-free directed component network graphs.

Experimental Data: The dataset of FP and FI comprises the data collected from real-world services, log files of simulation, and synthetic data. We use public QoS datasets WS-DREAM [32] in the experiments. WS-DREAM contains 339 * 5825 real-world QoS measurements, including both response time and throughput values, obtained from 339 users on 5,825 Web services. We used this dataset to calculate the failure probability of Web services from the perspective of individual users, which is used as the data resource of FP.

Since the failure impact of one service cannot be obtained directly from online services, we employ a simulation environment to conduct the experiments on FI. Application systems are built in our virtual cloud platform in the simulation. According to statistical FP distributions, we use fault injection tools to generate component failures intensively and record the failure events of components and applications. Fault injection tools can simulate failures according to predetermined distributions. We then use component invocation tracing to identify whether an application completes successfully [33], [34]. The invocation chain indicates the flow of business processing, including a sequential list of all components that are involved with processing the event. The invocation chain provides information about what components are used, times of use, and what component calls are made. Therefore, experiments using fault injection tests and component invocation tracing can make up for the lack of component failure data.
In addition, due to the sparsity of failures of real-world services, additional FP and FI data are synthesized as supplementary. The main focus of this part of experiment using synthetic data is not to evaluate the performance of proposed approach but rather to investigate the impact of various factors involved, so it is reasonable to use a synthetic dataset.

**Evaluation Method and Metric:** To improve availability of cloud applications with fault tolerance, there are two main tasks need to be done: 1) identifying the object of fault tolerance, and 2) selecting or designing the suitable fault tolerance strategy for them. This work focuses on the former part to develop a new component ranking approach, while the latter part is taken as the verification method to evaluate the proposed approach.

Among various fault tolerance strategies, we adopt redundancy in our experiments, the reasons are summarized as follows.

- Failures have different characteristics and causes. Meanwhile, failures can be classified into different groups according to their characteristics, such as duration (how long did a failure last), size (how many nodes failed together), type (hardware/software/network failure), cause (design flaw/defect/human misoperation). Some fault tolerant techniques have been developed for handling faults, however, a given technique generally addresses only one or a few types of system failure.

- Among multiple fault tolerant techniques, redundancy is a common and effective one. Various redundant fault tolerant methods can be developed based on different fault tolerant objects and technologies. For instance, failed task reexecution is an example of temporal redundancy, parallel execution with a replicated component is a type of special redundancy at the hardware level, N-version programming is a kind of special redundancy at the software level, and an additional check bit attached to a string of digital data is a form of special redundancy at the information level. The same objective of all these approaches is to achieve a higher reliability with redundant resources.

Therefore, we choose a typical fault tolerant strategy as the evaluation method, instead of to compare and select various fault tolerant strategies. Application designers also can choose other suitable fault tolerant strategies. After implementing parallel fault tolerant strategy, the component FP reduces to $FP(c_i) = FP(c_i) \prod_{k=1}^{n} FP(c_{i_k})$, where $n$ is the number of redundant components equipped for an important component. Here, we adopt $n = 1$, i.e., the failure probability of the optimized important component is $FP(c_i)^2$.

In this paper, we define the percentage improvement in application availability (PIAA) as the evaluation metric, which can be calculated by

$$PIAA = \frac{Av(opt) - Av(org)}{Av(org)} \times 100\%$$  \hspace{1cm} (9)

where $Av(org)$ and $Av(opt)$ represents the availability before and after optimization. A larger PIAA value indicates better performance.

$Av$ is the availability of an application (i.e. the probability that an application operates) under the common effect of all components. The original availability can be calculated by Eq. (10), while the optimized one can be calculated by Eq. (11).

$$Av(org) = 1 - \sum_{i=1}^{n} FP(c_i)FI(c_i)$$  \hspace{1cm} (10)

$$Av(opt) = 1 - \left( \sum_{r \in FT} FP^2(c_r)FI(c_r) \right) + \sum_{s \in FT^c} FP(c_s)FI(c_s)$$  \hspace{1cm} (11)

where $c_r (r \in FT)$ is the component with redundancy, while $c_s (s \in FT^c)$ is the component without redundancy.

**Compared Algorithms:** Two structure-based ranking algorithms are implemented and compared with FSCRank to study the performance of availability improvement:

- **FTCloud:** FTCloud is a component ranking framework for cloud applications presented in [24], which ranks the components by employing the structure information of the application. Fault tolerance strategies are employed to mask the faults of the topK percent components of the ranked list. The importance value in FTCloud is calculated as follows:

$$IV(c_i) = \frac{1 - d}{n} + d \sum_{k=1}^{n} IV(c_k)\omega_{ki}$$  \hspace{1cm} (12)

- **ROCloud:** ROClaud is an improved version based on FTCloud, which also takes the prior knowledge of component reliability properties into component ranking [25]. The importance value in ROClaud is calculated as follows:

$$IV(c_i) = \frac{1 - d}{n} f(c_i)p(c_i) + d \sum_{k=1}^{n} IV(c_k)\omega_{ki}$$  \hspace{1cm} (13)

For consistency of the form of equations, the variables with the same meaning but different names are replaced with the variable names defined in this paper, on the premise of remaining the original meanings of the importance value calculating equations of FTCloud and ROClaud.

The compared algorithms and our algorithm are all based on the PageRank algorithm. In PageRank-like algorithms, the parameter $d$ is used to balance the derived significance and the basic significance of the component itself. In related works [6], [24], [25], 0.85 has been widely used. Thus, we set this parameter as 0.85.

### 4.2 Performance Comparison

To evaluate and compare the performance of FTCloud, ROClaud, and FSCRank for cloud applications of different
scales with different failure tolerance ration (FTR), we conducted three groups of experiments with average FP equal to 0.01, 0.05 and 0.10 respectively. In each group of experiments, FTCloud, ROCloud, and FSCRank are applied on the structural graphs of applications with 100, 1000 and 10,000 component nodes, and FTR increasing from 10% to 50%.

FTR is the percentage of components with redundant resources of the total. For instance, FTR = k% means that the top k% most important components are equipped with redundant resources as the fault tolerance strategy. Each group is conducted for 100 times. The results of the average PIAA are plotted in the figures of Fig. 3. We use three different color to represent FTCloud (green), ROCloud (red) and FSCRank (blue). In addition, three different shades of the same color to illustrate one approach applied for applications with different scales. The result analyses are as follows:

- As the number of components increases, the PIAA decreases for FTCloud, ROCloud, and FSCRank with the same FTR values because a larger application system with a complex structure is affected more strongly by component failures than smaller ones. First, complex application systems are intrinsically hazardous and contain changing mixtures of failures. Second, joint faults are more likely to occur in a complex system because it involves more components and operations, where joint faults are more sufficient to create an accidental failure. Third, longer invocation chains increase the possibility of failure propagation and cascading failures.
- The performance of all three methods decreases as the average FP increases from 0.01 to 0.10, which indicates that the PIAA is sensitive to component failure probability.
- Since FTCloud simply takes the structure information as the basis of ranking, the corresponding results are the worst among all three methods. ROCloud improves the level of PIAA because this method considers the prior knowledge of the component reliability and structural information in the ranking process.
- FSCRank consistently outperforms FTCloud and ROCloud in all experimental settings because the structural differences between cloud applications and social networks are analyzed; in contrast, FTCloud and ROCloud fail to consider this information. The modification of the structure graph makes FSCRank more suitable and effective for cloud applications.

4.3 Impact of the Fault Tolerance Ratio

To provide a comprehensive analysis of the impact of the parameter FTR on the PIAA, we performed three groups of experiments with fixed component failure probability \( FP = 0.05 \), and different average component failure impact \( FI = 0.1, FI = 0.5, \text{ and } FI = 0.9 \). The FTR increases from 0 to 100% in each group. The node number of application is 1000. The results of PIAA with increasing FTR are plotted in Fig. 4. We use three different color to represent FTCloud (green), ROCloud (red) and FSCRank (blue). In addition, three different shades of the same color to illustrate one approach with different FI values. Figure 4 shows that:

- With increasing FTR, the performance of FTCloud, ROCloud, and FSCRank increase monotonically, which suggests that employing fault tolerance for more components with high-ranking scores results in greater improvement in application availability with all three approaches.
- Among the three approaches, FSCRank rises much faster than that of FTCloud and ROCloud. This indicates that relative to the other two methods, the availability of the cloud application can be improved much more by tolerating failures of the significant components selected by our component ranking approach.
- A further analysis of the increasing tendency of these three approaches shows that FSCRank has the highest growth rate when the fault tolerance percentage increases from 0 to 50%. A 20% improvement in application availability is obtained for each additional 10% fault tolerance. Thus, the improvement in application availability is almost that of full fault tolerance with only a 50% fault tolerance by FSCRank. In contrast, the same level of PIAA takes a higher percentage of the fault tolerance in FTCloud and ROCloud.
- This group of experiments suggests that FSCRank consistently outperforms FTCloud and ROCloud un-
Fig. 4 Average PIAA with increasing FTR.

under different values of fault tolerance, especially when there is a cost limit of fault tolerance. FSCRank achieves greater improvement in application availability with the same redundancy level.

4.4 Impact of the Component Availability Properties

In this section, we investigate the impact of the component availability properties on the performance of FTCloud, ROCloud and FSCRank. The component availability properties include component FP, FI and FI distribution. The synthetic data are used to evaluate the PIAA of these component ranking approaches thoroughly.

**Impact of the Component FP**: To study the impact of the component FP on the PIAA, we compared FTCloud, ROCloud and FSCRank under different FP settings. We first conducted tests with the average FP from 0 to 0.5, with a step value of 0.01. The value of FTR is set to 20%. Figure 5 shows the experimental results in terms of the PIAA.

- As the average FP increases, the PIAA gradually declines with all three approaches.
- The performance of FSCRank is much better than that of FTCloud and ROCloud when the average FP stays low. However, as the FP approaches 0.5, the performance of all three methods becomes basically the same because when the average component failure probability increases, the components with and without fault tolerance strategies are more likely to fail. In this situation, the average component failure probability increases greatly, which then leads to performance degradation.

**Impact of the Component FI**: We first reviewed the experimental results in Sect. 4.3 (Fig. 4) to study the impact of the component FI on the PIAA. We conducted three groups of tests with different average FI values.

- For the same FTR, the PIAA of each ranking method changes only slightly (less than 2%) under different FI value settings. This observation suggests that the change in average FI does not cause any substantial variation in performance. This conclusion can also be proved by the calculation equations of the importance value of all three methods.
- However, when we further explore the detailed experimental results, large differences in performance of each group are found, even though all groups of experiments are under the same average FI and FTR settings. We speculate that the reason the different distributions of component FI, although the average FI is the same.

**Impact of the FI Distribution**: To confirm the above speculation about the distribution of the component FI, we conducted another set of experiments on the FI with different standard deviations and the same mean values. The component FP is fixed to 0.05. Then, we take mean $FI = 0.5$ as an example; the range of the standard deviation $std$ of FI is in $[0.05, 0.40]$ with a step value is 0.05. The results are plotted as (a) to (h) in Fig. 6, which shows the following:

- The PIAA results of FTCloud, ROCloud and FSCRank all increase gradually as the std of FI increases, which
Fig. 6 Impact of the standard deviation of component FI on the PIAA.

indicates that the performance of ranking approaches improves when the range of FI is wider because they can identify the significant components more accurately. When the importance difference of component is significant, the FP decreases markedly by employing the fault tolerance strategy.

- The PIAA values of FTCloud, ROCloud and FSCRank are 8% (20% to 28%), 17% (22% to 39%), and 37% (31% to 68%), respectively, when the FTR is 20%.
- The increment of FTCloud is the smallest because it does not consider the component failure and failure impact in the process of component ranking. The important components are identified from the perspective of structural importance. The component FP works only in the phase when fault tolerance strategy is employed.
- Since ROCloud considers component failure and failure impact in both component ranking and fault tolerance employment, its performance is better than that of FTCloud.
- Similarly, FSCRank also considers component failure and failure impact on both of the phases of component ranking and fault tolerance employment. Furthermore, the improved structure makes FSCRank more sensitive to the component failure and failure impact because the weight of the component failure and failure impact works in both the release and retrieval of importance value for each component. Therefore, FSCRank achieves the best performance relative to FTCloud and ROCloud with the same settings.

4.5 Impact of the Leaf Node Ratio

The ratio of leaf nodes is a major difference between the structure graphs of cloud applications and social networks. Thus, to study the impact of the leaf node ratio, we generate a set of structure graphs with 1000 nodes and 2000 edges. We then increase the ratio of leaf nodes from 1% to 25% with a step value of 1%. The FTR is set as 20%. Figure 7 shows the experimental results:

- As the ratio of leaf nodes increases, the performance of FTCloud and ROCloud fluctuate in the ranges of [25%, 35%] and [30%, 45%], respectively; there are no prominent upward trends, which suggests that FTCloud and ROCloud are not sensitive to the ratio of leaf nodes.
- In contrast, the PIAA of FSCRank is the highest. In addition, the optimization level is proportional to the ratio of leaf nodes, which also turns out an upward trend, indicating that FSCRank is sensitive to the ratio of leaf nodes. That is because leaf nodes pass their entire important values to the buffer node, while non-leaf nodes release only part of their important values in each iteration. Thus, as LNR increases, the percentage of importance values passed to the buffer node rises, and then all nodes carve up this part importance value
of the buffer node in proportion to their failure influence. It means that the nodes with larger failure influence will obtain greater final importance values. This indicates FSCRank is more sensitive to component failure. Consequently, FSCRank achieves the best performance with the same FTR.

- For a fixed number of nodes and edges, different ratios of the leaf node imply different distributions of node degrees. As the leaf node ratio increases, the number of the nodes with zero out-degree increase, and thus the degree of nonleaf nodes increases. Consequently, the structural importance of the components changes. The increased PIAA of FSCRank indicates that it can capture the change in the leaf node ratio. Since the ratio of leaf nodes is one of the major differences between the structures of cloud applications and social networks, FSCRank is more suitable for component ranking and availability optimization of cloud applications than the contrast approaches.

5. Conclusion

In this paper, we propose the FSCRank component ranking approach for cloud applications. FSCRank integrates the component failure impact and the application structure information to evaluate component importance. By investigating structural characteristics of cloud applications, an iterative component ranking algorithm is presented. The introduction of the buffer node makes our algorithm more suitable and effective to rank components for cloud applications. We conduct extensive experiments on performance comparison and factor analysis. The experimental results show that FSCRank outperforms the contrast algorithms and achieves a more significant availability improvement. The quantitative analysis of the factors of availability improvement confirms the performance advantage and uniqueness of FSCRank.

Our ongoing and further research issues on component ranking for cloud applications include further analyzing the characteristics of real-world cloud applications and considering additional factors for component importance evaluation, and studying whether failure propagation exists in cloud application systems and affects component importance.

Acknowledgments

This work is supported by the National High Technology Development 863 Program of China under Grant No. 2013AA01A215.

References

[1] P. Mell and T. Grance, “The NIST definition of cloud computing,” NIST Special Publication, 2011.
[2] M. Hogan, F. Liu, A. Sokol, and J. Tong, “Nist cloud computing standards roadmap,” NIST Special Publication, vol.35, pp.6–11, 2011.
[3] P.T. Endo, M. Rodrigues, G.E. Gonçalves, J. Kelner, D.H. Sadok, and C. Carescu, “High availability in clouds: systematic review and research challenges,” J. Cloud Computing, vol.5, no.1, p.16, Oct. 2016.
[4] S. Hariri, A. Choudhary, and B. Sarikaya, “Architectural support for designing fault-tolerant open distributed systems,” Computer, vol.25, no.6, pp.50–62, June 1992.
[5] F.C. Meng, “Some further results on ranking the importance of system components,” Reliability Engineering & System Safety, vol.47, no.2, pp.97–101, 1995.
[6] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web,” Tech. Rep., Stanford InfoLab, 1999.
[7] J. Leskovec, A. Rajaraman, and J.D. Ullman, Mining of massive datasets, Cambridge university press, 2014.
[8] 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,” Tech. Rep., UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.
[9] P. Fergusson and G. Huston, Quality of Service: Delivering QoS on the Internet and in corporate networks, Wiley New York, 1998.
[10] P. Patel, A.H. Ranabahu, and A.P. Sheth, “Service level agreement in cloud computing,” Knoesis Publications, 2009.
[11] L. Qian, Z. Luo, Y. Du, and L. Guo, “Cloud computing: An overview,” IEEE Int. Conf. Cloud Computing, vol.5931, pp.626–631, Springer, 2009.
[12] M. Blanke, M. Staroswiecki, and N.E. Wu, “Concepts and methods in fault-tolerant control,” Proc. 2001 American Control Conference, pp.2606–2620, IEEE, 2001.
[13] W. Kuo and X. Zhu, Importance measures in reliability, risk, and optimization: principles and applications, John Wiley & Sons, April 2012.
[14] Z.W. Birnbaum, “On the importance of different components in a multicomponent system,” Tech. Rep., Washington Univ Seattle Lab of Statistical Research, 1968.
[15] W. Kuo and M.J. Zuo, Optimal Reliability Modeling: Principles and applications, John Wiley & Sons, 2003.
[16] F.P. Brooks Jr., “Three great challenges for half-century-old computer science,” J. ACM (JACM), vol.50, no.1, pp.25–26, Jan. 2003.
[17] L.C. Freeman, “Centrality in social networks conceptual clarification,” Social networks, vol.1, no.3, pp.215–239, 1978.
[18] P. Bonacich, “Factoring and weighting approaches to status scores and clique identification,” J. Mathematical Sociology, vol.2, no.1, pp.113–120, 1972.
[19] L.C. Freeman, “A set of measures of centrality based on betweenness,” Sociometry, vol.40, no.1, pp.35–41, March 1977.
[20] C.C. Hsu, Y.A. Lai, W.H. Chen, M.H. Feng, and S.D. Lin, “Unsupervised ranking using graph structures and node attributes,” Proc. Tenth ACM Int. Conf. Web Search and Data Mining, pp.771–779, ACM, 2017.
[21] J.M. Kleinberg, “Authoritative sources in a hyperlinked environment,” J. ACM (JACM), vol.46, no.5, pp.604–632, Sept. 1999.
[22] W. Xing and A. Ghorbani, “Weighted pagerank algorithm,” Proceedings, Second Annual Conference on Communication Networks and Services Research, 2004, pp.305–314, IEEE, 2004.
[23] L. Zhang, T. Qin, T.Y. Liu, Y. Bao, and H. Li, “N-step pagerank for web search,” European Conference on Information Retrieval, vol.4425, pp.653–660, Springer, 2007.
[24] Z. Zheng, T.C. Zhou, M.R. Lyu, and I. King, “Component ranking for fault-tolerant cloud applications,” IEEE Trans. Services Computing, vol.5, no.4, pp.540–550, 2012.
[25] W. Qiu, Z. Zheng, X. Wang, X. Yang, and M.R. Lyu, “Reliability-based design optimization for cloud migration,” IEEE Trans. Services Computing, vol.7, no.2, pp.223–236, April–June 2014.
[26] S.K. Garg, S. Versteeg, and R. Buyya, “A framework for ranking of
cloud computing services,” Future Generation Computer Systems, vol.29, no.4, pp.1012–1023, June 2013.

[27] N. Somu, K. Kirthivasan, and S.S. VS, “A computational model for ranking cloud service providers using hypergraph based techniques,” Future Generation Computer Systems, vol.68, pp.14–30, March 2017.

[28] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, “Documenting software architectures: views and beyond,” Proc. 25th Int. Conf. Software Engineering, pp.740–741, IEEE Computer Society, 2003.

[29] M. Yu-Tao, H. Ke-Qing, L. Bing, and L. Jing, “Empirical study on the characteristics of complex networks in networked software,” J. Software, vol.22, no.3, pp.381–407, 2011.

[30] C.R. Myers, “Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs,” Physical Review E, vol.68, no.4, p.046116, Oct. 2003.

[31] A. Hagberg, P. Swart, and D. S Chult, “Exploring network structure, dynamics, and function using networkx,” Tech. Rep., Los Alamos National Lab. (LANL), Los Alamos, NM (United States), 2008.

[32] Z. Zheng and M.R. Lyu, “Collaborative reliability prediction for service-oriented systems,” Proc. 32nd IEEE/ACM Int’l Conf. Software Engineering (ICSE’10), pp.35–44, 2010.

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

[34] A. Waqas, Z.M. Yusof, and A. Shah, “Fault tolerant cloud auditing,” 2013 5th Int. Conf. Information and Communication Technology for the Muslim World (ICT4M), pp.1–5, IEEE, 2013.

Na Wu is currently a Ph.D. candidate at the fault tolerance and mobile computing research center, Harbin institute of Technology. Her advisor is Prof. Decheng Zuo. She received B.S. and M.E. degree from Harbin Institute of Technology in 2007 and 2011. Her research interests include fault tolerance, availability optimization and resource scheduling in cloud computing.

Decheng Zuo is a professor in the school of computer science and technology, Harbin Institute of Technology. He received Ph.D. degree in computer science and technology from Harbin Institute of Technology in 2000. His research interests include parallel computing and architecture, fault tolerant computer, computer system architecture evaluation theory and technology. He is a member of the expert committee for Information Field, National High-tech R&D Program of China (863 Program).

Zhan Zhang is an associate professor in the school of computer science and technology, Harbin Institute of Technology. He received Ph.D. degree in computer science and technology from Harbin Institute of Technology in 2008. His research interests include fault tolerant computer, wearable computer and system evaluation theory and technology.

Peng Zhou is currently a joint Ph.D. student of Harbin Institute of Technology, China, and University Clermont Auvergne, France. He received B.S. and M.E. degree from Harbin Institute of Technology in 2010 and 2012. His research area is the dependable self-managing large scale CPS and edge computing system design and evaluation.

Yan Zhao is currently a Ph.D. candidate at the fault tolerance and mobile computing research center, Harbin Institute of Technology. He received B.S. degree from Harbin Institute of Technology, China, in 2014, and M.E. degree from University Clermont Auvergne, France, in 2016. His research interests include the theory and technology in resource scheduling, cloud computing, and service evaluation.