ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

Volume 3, Special Issue 3, March 2014

2014 International Conference on Innovations in Engineering and Technology (ICIET'14)

On 21st& 22nd March Organized by

K.L.N. College of Engineering, Madurai, Tamil Nadu, India

Improving Resilience of Cloud Application Using Ranking Technique

Anand M and KannigaDevi R

PG Student, Dept of Computer Science Engg., Kalasalingam University, Krishnankoil, Virudhunagar, India.

Assistant Professor, Dept of Computer Science Engg., Kalasalingam University, Krishnankoil, Virudhunagar, India.

Abstract - Cloud computing is a general term for anything that involves delivering hosted services over the Internet. A Fault tolerance is a setup or configuration that prevents a computer or network device from failing in the event of an unexpected problem or error. In this project work, we propose a model to analyze an optimal fault tolerant strategy to improve the resilience of cloud applications. The cloud applications are usually large scale and include a lot of distributed cloud components. Building highly efficient cloud applications is a challenging and critical research problem. To attack this challenge a component ranking frame work, named FTCloud is used for building fault tolerant cloud applications. First extract the components from the cloud application. Then, rank the critical components using the significance value. After the component ranking phase, an algorithm is projected to automatically conclude an optimal fault-tolerance strategy for the significant cloud components. Thereby, resilience of cloud application can be improved.

Keywords — Cloud application, component ranking technique, fault tolerance

I.INTRODUCTION

Cloud is a general term for anything that involves delivering hosted services over the Internet. It is getting popular in recent years. The software systems in the cloud (named as cloud applications) typically involve multiple cloud components communicating with each of them [1]. Basically cloud applications are usually huge and very complex. Regrettably, the reliability of the cloud applications is still far from perfect in real life. The requirement for highly reliable cloud applications is becoming unprecedented strong. Building highly efficient clouds becomes a critical, challenging, and urgently required research problem. The trend toward large-scale complex cloud applications makes developing fault-free systems by only employing fau lt-prevention techniques and fault-removal techniques exceedingly difficult. Another approach for building efficient systems, software fault tolerance [20], makes the system more robust by faults masking without removing it. One of the most well known software fault tolerance techniques is to employ functionally equivalent yet independently designed components to tolerate faults [5]. Due to the cost of developing and maintaining redundant components, software fault tolerance is usually only employed for critical systems. Different from traditional software systems, there are a lot of redundant resources in the cloud environment, making software fault tolerance a possible approach for building highly reliable cloud applications. Since cloud applications usually involve a large number of components, it is still too expensive to provide alternative components for all the cloud components. Moreover, there is probably no need to provide fault tolerance mechanisms for the non critical components, whose failures have limited impact on the systems. To reduce the cost so as to develop highly reliable cloud applications within a limited budget, a small set of critical components needs to be identified from the cloud applications. By tolerating faults of a small part of the most important cloud components, the cloud application reliability can be greatly improved. Based on this idea, we propose FTCloud, which is a component ranking framework for building fault tolerant cloud applications. the optimal fault-tolerance strategies for these significant components automatically. FTCloud can be employed by designers of cloud applications to design more reliable and robust cloud applications efficiently and effectively.

Contribution of this paper:

. This paper identifies the critical problem of locating significant components in complex cloud applications and proposes a ranking-based framework, named FTCloud, to build fault-tolerant cloud applications. We first propose ranking algorithms to identify significant components from the huge amount of cloud components. Then, we present an optimal fault-tolerance strategy selection algorithm to determine the most suitable fault-tolerance strategy for each significant component. We consider FTCloud as the first ranking-based framework for developing fault-tolerant cloud applications.

We provide extensive experiments to evaluate the impact of significant components on the reliability of cloud applications.

II. RELATED WORK

Component ranking is an important research problem in cloud computing [41], [42]. The component ranking approaches of this paper are based on the intuition that components which are invoked frequently by other important components are more important. Similar ranking approaches include Google Page rank [7] (a ranking algorithm for web page searching) and SPARS-J [16] (software product retrieving system for Java). Different from the Page Rank and SPARS-J models, component invocation frequencies as well as component characteristics are explored in the approaches. Moreover, the target of approach is identifying significant components for cloud applications instead of web page searching (Page Rank) or reusable code searching (SPARS-J).

Cloud computing [3] is being popular. The works have been done on cloud computing, including identifies the critical to address fault tolerant strategy [32], identifies the effects of failures on user's applications, and surveying fault tolerance solutions corresponding to each class of failures [9], Too inadequate or too expensive to fit their individual requirements [34], etc. In recent years, a great number of research efforts have been performed in the area of service component selection and composition [30]. Various approaches, such as QoS-aware middle ware [38], adaptive service composition [2], and efficient service selection algorithms [37], have been proposed. Some recent efforts also take subjective information (e.g., provider reputations, user requirements, etc) to enable more accurate component selection [27]. Instead of employing non functional performance (e.g., QoS values) or functional capabilities, the approaches rank the cloud components considering component invocation relationship, invocation frequencies, and component cha-racteristics.

III. SYSTEM ARCHITECTURE

Fig.1shows the system architecture of the fault-tolerance framework (named FTCloud), which includes two parts: 1) ranking and 2) optimal fault-tolerance selection. The procedures of FTCloud are as follows:

1. The system designer provides the initial architecture design of a cloud application to FTCloud. A component extraction can be done for the cloud application based on the weight value.

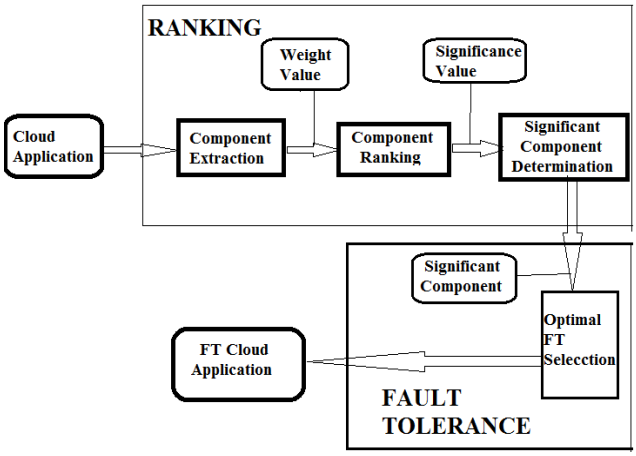

Figure:1- System Architecture

2. Significance values of the cloud components are calculated by employing component ranking algorithms. Based on the significance values, the components can be ranked.
3. The most significant components in the cloud application are identified based on the ranking results.
4. The performance of various fault-tolerance strategy candidates is calculated and the most suitable fault-tolerance strategy is selected for each significant component.
5. The component ranking results and the selected fault-tolerance strategies for the significant components are returned to the system designer for building a reliable cloud application.

IV. PROPOSED WORK

IV.I .SIGNIFICANT COMPONENT RANKING:

The target of significant component ranking algorithm is to measure the importance of cloud components based on available information (e.g., application structure, component invocation relationships, component characteristics, etc.).The significant component ranking includes three steps (i.e., component Extraction, component ranking, and significant component determination)

IV.I.I. Component Extraction:

A cloud application can be modeled as a weighted, where a node c_i represents a component and a directed

.

edge e_{ij} from node c_i to node c_j represents a component invocation relationship, i.e., c_i invokes c_j. Each node c_i has a nonnegative significance value $V(c_i)$, which is in the range of (0,1). Each edge e_{ij} has a nonnegative weight value $W(e_{ij})$, which is in the range of [0,1]. The weight value of an edge e_{ij} can be calculated by

$$W(e_{ij}) = frq_{ij} / \sum_{j=1}^{n} frq_{ij} \quad (1)$$

Where frq_{ij} is the invocation frequency of component c_j by component c_i, n is the number of components, and frq_{ij} = 0 if component c_i does not invoke c_j. In this way, the edge e_i has a larger weight value if component c_j is invoked more frequently by component c_i compared with other components invoked by c_i. For a component extraction, C which contains n components, an n * n transition probability matrix W can be obtained by employing (1) to calculate the invocation weight values of the edges. Each entry w_{ij} in the matrix is the value of $W(e_{ij})$. w_{ij} = 0 if there is no edge from c_i to c_j, which means that c_i does not invoke c_j. If a component does not invoke itself, w_{ii} = 0. Otherwise, the value of w_{ii} can be calculated by (1). In the case that a node c_i has no outgoing edge, w_{ij} = 1\ n. For i, a single component of an application, C can be obtained by weight of an edge, $W(e_{ij})$

$$C = \forall i. \sum_{j=1}^{n} W(e_{ij}) \quad (2)$$

IV.I.II. Component Ranking:

Based on the component extraction, a component ranking algorithms, named as FTCloud is proposed in this section. It employs the system structure information (i.e., the component invocation relationships and frequencies) for making component ranking and also considers the component characteristics (i.e., critical components or noncritical components) for making component ranking.Figure: 2 shows the critical and non crical components based on significance value.

IV.I.III. FTCloud-Based Component Ranking:

In a cloud application, some of the components are considered to be more important which are frequently invoked by a lot of other components. Since their failures will have greater impact on the whole system. Probably, the significant components in a cloud application are the ones which have many invocation links coming in from the other important components. Inspired by the PageRank algorithm [7], we propose an algorithm to calculate the significance values of the cloud components applying the component invocation relationships and frequencies. The procedure of FTCloud-based component ranking algorithm is shown in the following steps:

1. Randomly assign initial numerical scores between 0 and 1 to the components
2. Compute the significance value for a component c_i by:

$$V(c_i) = (1-d)/n + d\sum_{k} \epsilon N(c_i) V(c_k) W(e_{ki}) \quad (3)$$

Where n is the number of components and $N(c_i)$ is a set of components that invoke component c_i. The parameter d ($0 \leq d \leq 1$) in (3) is employed to adjust the significance values derived from other components, so that the significance value of c_i is composed of the basic value of itself (i.e.,$(1-d)/n$) and the derived values from the components that invoked c_i. By (3), a component c_i has larger significance value indicating that component c_i invoked by a lot of other significant components frequently.

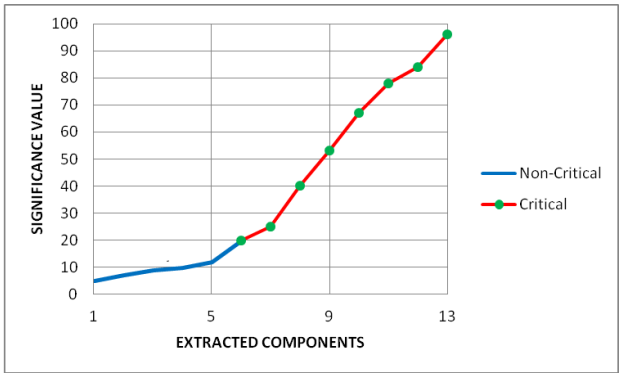

Figure:2- Significant Component Identification

IV.I.IV. Significant Component Determination

The components of cloud application can be ranked based on obtained significance value and the top k ($1 \leq k \leq n$) most significant components can be returned to the cloud application's designer. After that significant components can be obtained by the designer of cloud application at a time of architecture design and can employ various techniques to improve the resilience of the cloud application.

IV.II. FAULT-TOLERANCE STRATEGY SELECTION

IV.II.I. Fault-Tolerance Strategies

Software fault tolerance is widely adopted to increase the overall system reliability in cloud applications. Applying functionally equivalent components to tolerate component failures, thereby resilience can be improved. There are three most common fault-tolerance methods with formulas to find out the failure probabilities of the each fault-tolerant method. The failure probability should be within the range of [0,1].

IV.II.II. Recovery block(RB).

Execute a component, if fails through acceptance test, then try a next alternate component. Order the different

component according to reliability. Checkpoints needed to provide valid operational state for subsequent versions (hence, discard all updates made by a component). Acceptance test needs to be faster and simpler than actual code. A recovery block fails only if all the redundant components fail.

The failure probability f of a recovery block :

$$f = \prod_{i=1}^{n} f_i \quad (4)$$

Where n is the number of alternate components and fi is the failure probability of the ith component.

IV.II.III. N-version programming (NVP).

N-version programming, also known as multi version programming, all versions designed to satisfy same basic requirement. Decision of output comparison based on voting. Different teams build different versions to avoid correlated failures. When applying the NVP approach to the cloud application's component, implement the equivalent function of cloud components should be alone and involved in parallel and then final result is determined by majority voting. It will fail only if more than half of the redundant components stop working. The failure probability fi of an NVP module can be computed.

$$f = \sum_{i=(n+1)/2}^{n} F_i \quad (5)$$

Where n is the number of equivalent components (n is usually an odd number in NVP)

IV.II.IV. Parallel.

Parallel strategy invokes all the n functional equivalent components in parallel and the first returned response will be employed as the final decision. It fails only if all the alternate components stop working. The failure probability f of a parallel module can be computed by

$$f = \prod_{i=1}^{n} f_i \quad (6)$$

Where n is the number of alternate components and fi is the failure probability of the ith component.

Different features of fault tolerance strategies, the response time of RB and NVP strategies is not good compared with Parallel strategy in performance wise, while Parallel strategy employs the first returned response as the final decision. The required resources of RB are much lower than those of NVP and Parallel since parallel component invocations consume a lot of networking and computing resources. RB, NVP, and Parallel strategies can tolerate crash faults. NVP can also mask value faults (e.g., data corruption), the final results in NVP can be determined through majority voting.

IV.II.V. Optimal FT Strategy Selection

The fault-tolerance strategies have a number of variations based on different setups. Fault tolerance method variations are applied for each and every single significant component in a cloud application and the optimal one need to be identified. For each significant component that requires a fault tolerance strategy, the designer can specify constraints (e.g., response time of the component has to be smaller than 1,000 milliseconds, etc.).Response time and cost are the two user constraints should be noted. The optimal fault-tolerance method selection problem for a cloud component with user constraints can then be formulated mathematically. First, the candidates which cannot meet the user constraints are not include. Then the fault-tolerance candidate with the best failure probability performance will be selected as the optimal strategy for component i. By the above approach, the optimal fault-tolerance method, gives the best failure probability performance and fulfill all the user constraints.

To identify optimal FT Strategy Selection:

Input: si, ti, and fi values of candidates; user constraints u1, u2;
Output: Optimal candidate index p
m: number of candidates;
for (i =1; i ≤ m; i++) do
if (si≤ u1&&ti ≤u2) then
vi = fi;
end
end
if none of the candidate meet user constraints after that Throw exception;
end
Select vx which has minimal value from all the vi;
P= x;

Algorithm1. Optimal FT Strategy Selection

The above algorithm identifies optimal FT strategy selection for each of significant component. Based on result the designer have to apply the identified FT strategy, thus resilience of the cloud application can be improved.

V. EXPERIMENT

V.I. Experimental Setup

The significant component ranking algorithms are implemented by java language using cloudsim tool based on hundred nodes. To find out the performance of reliability increment, we compare four approaches, which are as follows:

No FT: No fault-tolerance strategies are employed for the components in the cloud application.

Random FT. Fault-tolerance strategies are employed to mask faults of K percent components, those components are randomly selected.

FTCloud: Fault-tolerance strategies are employed to mask faults of the Top-K percent important components (using significance value). The components are ranked based on the structural information of the cloud application.

AllFT: Fault-tolerance strategies are employed for all the cloud components.

VI. COMPONENT FAILURE PROBABILITY IMPACT

To learn the impact of the system resilience on cloud application, we compare RandomFT and FTCloud under probability set from 0.1 to 1 percent with a step value of 0.1 percent. Thousands node are taken for this execution. Implementation result shows cloud application failure probabilities Fig. 3.

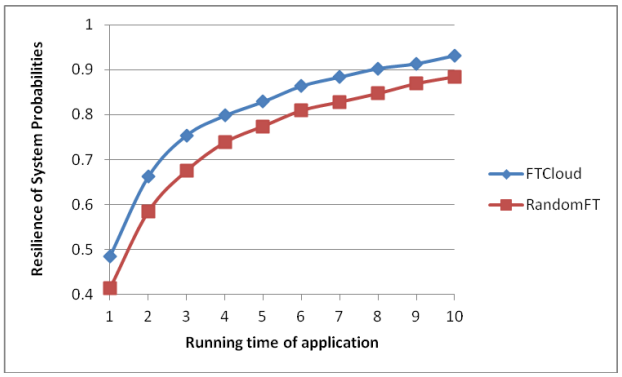

FIGURE: 3- The impact of the system resilience

Fig. 3 explains

FTCloud outperform RandomFT in all the application running time settings from 1 percent constantly as shown in Fig. 3

The system resilience probabilities of the two methods become larger, when application runs. To build highly reliable cloud applications, then a larger number of significant components are needed.

The application failure probability of FTCloud approach decreases much faster than that of RandomFT, representing that have a more efficient use of the redundant components than RandomFT, by the increase the selection of more significant components.

VII. CONCLUSION AND FUTURE WORK

This paper proposes a component ranking framework cloud application's component. The significance values of these components, how often the current component is called by other components, and the component quality.

After determine the significant components, we suggest an optimal fault-tolerance strategy selection algorithm to afford optimal fault-tolerance strategies to the significant components automatically, based on the user limits. The implementation results display the FTCloud approaches.

The current FTCloud framework can be engaged to bear crash and significance faults. In the future, we will examine more types of faults, such as Byzantine faults. Various types of fault-tolerance mechanisms can be extra into FTCloud framework effortlessly without Basic changes. We will also examine additional component ranking algorithms and add them to the FTCloud framework. Moreover, we will expand and practical FTCloud framework to other component-based systems.

In this paper, we only learn the most delegate type of software component extraction, i.e., weight value of an edge. as different applications may have dissimilar system structures, we will examine more types of component models in future work.

The future work also includes

- Allow more factors (such as invocation delay, output, etc.) when computing the weights of invocations links;
- Examining the component consistency itself moreover the invocation structures and invocation frequencies;
- More investigational testing on real-world cloud applications.
- more examination on the component malfunction correlations; and
- More new studies on collision of incorrectness of prior wisdom on the invocation frequencies and essential components.

REFERENCE

[1] "cloud computing in wikipedia," http://en.wikipedia.org/wiki/cloud_computing, 2012.
[2] d. Ardagna and b. Pernici, "adaptive service composition in flexible processes," june 2007.
[3] m. Armbrust et al., "a view of cloud computing," 2010.
[4] m. Armbrust et al., "above the clouds: a berkeley view of cloud-computing," 2009.
[5] a. Avizienis, "the methodology of n-version programming," software fault tolerance, 1995.
[6] v. Batagelj and a. Mrvar, "pajek - program for large network analysis," 1998.
[7] s. Brin and l. Page, "the anatomy of a large-scale hypertextual web search engine," 1998.
[8] m. Creeger, "cloud computing: an overview,", june 2009.
[9] Huang and Abraham, "Providing Reliability as an Elastic Service in Cloud Computing", 2011.
[10] a.p.s. de moura, y.-c. Lai, and a.e. motter, "signatures of small- world and scale-free properties in large computer programs," 2003.
[11] c.-l. Fang, d. Liang, f. Lin, and c.-c. Lin, "fault tolerant web services," 2007.
[12] s.s. gokhale and k.s. trivedi, "reliability prediction and sensitivity analysis based on software architecture," 2002.
[13] s. Gorender, r.j. de araujo macedo, and m. Raynal, "an adaptive programming model for fault-tolerant distributed computing," jan.-mar. 2007.
[14] a. Goscinski and m. Brock, "toward dynamic and attribute-based publication, discovery and selection for cloud computing,", 2010.
[15] d. Hyland-wood, d. Carrington, and y. Kaplan, "scale-free nature of java software package, class and method collaboration graphs," 2009.
[16] k. Inoue, r. Yokomori, t. Yamamoto, m. Matsushita, and s. Kusumoto, "ranking significance of software components based on use relations," mar. 2009.
[17] k. Kim and h. Welch, "distributed execution of recovery blocks: an approach for uniform treatment of hardware and software faults in real-time applications," may 1989.
[18] j. Laprie, j. Arlat, c. Beounes, and k. Kanoun, "definition and analysis of hardware- and software-fault-tolerant architectures," july 1990.
[19] w. Li, j. He, q. Ma, i.-l. Yen, f. Bastani, and r. Paul, "a framework to support survivable web services," 2005.
[20] m.r. lyu, software fault tolerance, wiley, 1995.
[21] m.r. lyu, handbook of software reliability engineering. Mcgraw-hill, 1996.
[22] e. Maximilien and m. Singh, "conceptual model of web service reputation,"2002.

[23] m.g. merideth, a. Iyengar, t. Mikalsen, s. Tai, i. Rouvellou, and p. Narasimhan, "thema: byzantine-fault-tolerant middleware forweb-service applications," 2005.
[24] s.l. pallemulle, h.d. thorvaldsson, and k.j. goldman, "byzantine fault-tolerant web services for n-tier and service oriented architectures," 2008. Zheng et al.: component ranking for fault-tolerant cloud applications.
[25] b. Randell and j. Xu, "the evolution of the recovery block concept," software fault tolerance, m.r. lyu, , wiley,1995.
[26] p. Rooney, "microsoft's ceo: 80-20 rule applies to bugs, not just features," oct. 2002.
[27] s. Rosario, a. Benveniste, s. Haar, and c. Jard, "probabilistic qos and soft contracts for transaction-based web services orchestrations," oct. 2008.
[28] j. Salas, f. Perez-sorrosal, m. Patin˜ o-marti´nez, and r. Jime´nez-peris, "ws-replication: a framework for highly available web-services," 2006.
[29] g.t. santos, l.c. lung, and c. Montez, "ftweb: a fault tolerant infrastructure for web services," 2005.
[30] q.z. sheng, b. Benatallah, z. Maamar, and a.h. ngu, "configurable Composition and adaptive provisioning of web services," jan.-mar.2009.
[31] g.-w. Sheu, y.-s. Chang, d. Liang, s.-m. Yuan, and w. Lo, "afault-tolerant object service on corba," 1997.
[32] Jing Deng and Wang et al, "Fault-Tolerant and Reliable Computation in Cloud Computing", 2011.
[33] w.-t. Tsai, x. Zhou, y. Chen, and x. Bai, "on testing and evaluating service-oriented software," aug. 2008.
[34] Mei et al and cho li wang,"Web product ranking using opinion mining", 2011.
[35] g. Wu, j.wei, x. Qiao, and l. Li, "a bayesian network based qos assessment model for web services," 2007.
[36] s.m. yacoub, b. Cukic, and h.h. ammar, "scenario-based reliability analysis of component-based software," 1999.
[37] t. Yu, y. Zhang, and k.-j. Lin, "efficient algorithms for web services selection with end-to-end qos constraints," 2007.
[38] l. Zeng, b. Benatallah, a.h. ngu, m. Dumas, j. Kalagnanam, and H. Chang, "qos-aware middleware for web services composition," may2004.
[39] z. Zheng and m.r. lyu, "a distributed replication strategy evaluation and selection framework for fault tolerant web services," 2008.
[40] z. Zheng and m.r. lyu, "a qos-aware fault tolerant middleware for dependable service composition," 2009.
[41] z. Zheng, y. Zhang, and m.r. lyu, "cloudrank: a qos-driven component ranking framework for cloud computing," 2010.
[42] z. Zheng, t.c. zhou, m.r. lyu, and i. King, "ftcloud: a ranking-based framework for fault tolerant cloud applications," 2011