Fault Localization in Cloud using Graph Theoretic Techniques

Narayanaa S R, Sivaranjan M and Dr. Lekshmi R S

PSG College of Technology, Coimbatore

Keywords: Fault Localization, Centrality Measures, Cloud Computing

Abstract

Fault localization is an imperative method in fault tolerance in distributed environment which designs a blueprint for continuing the ongoing process even when one or many modules are non-functional. Visualising a distributed environment as a graph allows us to introduce probabilistic weights to both edges and nodes that cause the faults. With multiple modules like databases, run-time cloud etc. making up a distributed environment and extensively, a cloud environment, we aim to solve the problem of optimally and accurately performing fault localization in distributed environment by modifying Graph optimization approach to localization and centrality, specific to fault graphs.

1 INTRODUCTION

Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. Large clouds, predominant today, often have functions distributed over multiple locations from central servers. Cloud computing delivers three types of computing resources, namely hardware, software, and software development framework, also known as platform. The cloud computing, as a fast advancing technology, is increasingly being used to host many business or enterprise applications. However, the extensive use of the cloud-based services for hosting business or enterprise applications leads to service reliability and availability issues for both service providers and users. Several types of faults may occur in the cloud environment leading to failures and performance degradation [1].

Failures lead to system breakdown or shut down of a system. However, any distributed computing architecture is characterized by the notion of partial failures. A fault may occur in any constituent node, process or network component. This leads to a partial failure and consequently, performance degradation instead of a complete breakdown. Fault tolerance in a system is the capability of a system to continue performing its anticipated function regardless of faults. This way, a fault tolerant system is expected to be reliable and operate successfully in the case of a failure or breakdown. In case of a fault occurring from any particular node, a fault tolerant system must be able to identify potential vulnerable modules to replicate their functionality to prevent complete failure of the system.

Centrality measures in a graph indicate influential nodes and links in the network. In most cases, they show the relative significance of nodes in the system. Researchers have developed many algorithms for identifying important nodes in a network. The basic and locally computed centrality is degree centrality. Other commonly used metrics are betweenness centrality, closeness centrality, PageRank, HITS eigenvector centrality [2], which are more relevant in single-layer networks, limiting their applications in multi-layer networking systems. Among these, PageRank is a popular and frequently used metric, designed for ranking of web pages, it is widely applied in other applications also.

State-of-the-art fault localization approaches rely on long training sessions based on simulated faults, which is an expensive practice, since it is hard to inject multiple classes of faults in several machines with different access controls, and iterate this process while the system evolves [3].

In this paper, we propose a method that help us accurately identify important modules in cloud and localize it in case of a breakdown or failure, triggered by itself or another component it depends on.

The contributions of this project work are: (i) an efficient approach to accurately identify highly important and vulnerable components (ii) an efficient method to localize the fault triggering components (iii) an experimental evaluation of the suggested method for identifying and locating different combinations of faults arising from different modules.

The paper is organized as follows. Section II overviews the existing work on fault localization techniques. Sections III represent the centrality measure based localization method. Section IV illustrates the experimental results of the approach. Section V summarizes the main contribution of the paper.

2 RELATED WORK

There are also three types of failure: Hardware, Virtual Machine (VM) and Application failure. But there is more than one fault such as a permanent fault that means there is a mistake in the component. A transient fault means fault takes some time until all component functionality restored [11].

Any fault occurs during the execution leads to re-running the application from beginning in the same machine. Alternatively, add a new machine that will cost time, money and resources. Therefore, we need to minimize failure impact on the system and application execution. The failures should be handled by using Fault tolerance techniques. Fault tolerance ensures more availability and reliability of services using virtual machine copies during application execution. Reliability of system is referred to the ability of the machine to complete the execution of application successfully understated condition.

To handle the fault that may occur there are different types of techniques of fault tolerance that classified in two categories:

2.1 Proactive Techniques

In Proactive Fault Tolerance, the problems are foreseen before it occurs and is avoided based its influence on the task. Also, proactive fault tolerance prevents the tasks and VMs from failing and makes sure the task is executed correctly. One of the methods in Proactive Fault Tolerance is preemptive migration, which uses a control system where the tasks are proctored. When a task is found to be vulnerable to exit, it will be shifted to another virtual machine instance. The disadvantage in this method is the wait time for calling another machine [4]. One of the proactive methods for fault tolerance is PCFT (Proactive Co-ordinated fault tolerance), a fault tolerating method that accepts a virtual machine harmonized and coordinated method to look for a deteriorating physical machine caused by a phenomenon called hypervisor ageing, in the data center and then subsequently performs virtual machine migration to some optimal target automatically.

2.2 Reactive Techniques

In Reactive Fault Tolerance, once the fault done in computing node or task it begins to reduce its efforts as far as possible to ensure the task completed its job. Steps involved in reactive fault tolerance include (a) Replication : a technique used to replicate the task on more than one resource, (b) Task resubmission: when the task fails it will run again in either the same node or to a different resource for execution from the beginning of the task, where this increases the execution time over the expected time because of the repetition of the task (c) Retry: re-execution of the failed task in the same resource, but it consumes the execution time like the re-submission technique, (d) Check pointing or Restart: It takes snapshots at different times during task execution to re-running the task when failed from the last point, not from the beginning [5].

Other methods include meta-heuristic based method, based on heuristic, mainly used to direct the search procedure where the objective is to effectively examine the search space to locate the near optimal solutions.

3 PROPOSED MODEL

Proposed centrality measure based fault localization method, assigns the importance of modules in a given cloud architecture with the rank values of different faults associated with a component obtained from the fault graph model generated. Computation is based on iterations over fault nodes with weights on edges and nodes calculated with probabilistic values. The following subsections describe the introduction of faults in cloud architecture that are covered and the proposed centrality measure and its computational procedure.

3.1 Theoretical Background

In this section, we discuss some basic concepts about the types of faults in cloud architecture and centrality measures in graph.

3.1.1 Cloud Computing Infrastructure

Cloud computing infrastructure includes the computers, network facilities, storage devices, and other connected components necessary for offering cloud computing resources and services to users. These hardware components are located within enterprise data centers. These encompass solid-state drives, multi-core servers and hard disk drive offering stable storage and network devices, such as firewalls, switches, and routers; all on a large scale [6].

Various components in a distributed system	
Servers	The physical machines that act as host machines for one or more virtual machines.
Virtualization	Technology that abstracts physical components such as servers, storage, and networking and provides these as logical resources.
Storage	In the form of Storage Area Networks (SAN), network attached storage (NAS), disk drives etc. Along with facilities as archiving and backup.
Network	To provide interconnections between physical servers and storage.
Management	Various software for configuring, management and monitoring of cloud infrastructure including servers, network, and storage devices.
Security	Components that provide integrity, availability, and confidentiality of data and security of information, in general.

Table 1. Various components in a distributed system

There are many factors that could trigger a fault including data corruption, hardware or software or network outages, unreliable clocks, method failure, parity error, head crash, denial of service, disk full, etc. Faults in general are classified into below mentioned categories :

Network fault: Since cloud computing resources are accessed over a network (Internet), a predominant cause of failures in cloud computing are the network faults. These faults may occur due to partitions in the network, packet loss or corruption, congestion, failure of the destination node or link, etc.

Physical faults: These are faults that mainly occur in hardware resources, such as faults in CPUs, in memory, in storage, failure of power etc.

Process faults: These faults may occur in processes because of resource shortage, bugs in software, incompetent processing capabilities, etc.

Service expiry fault: If a resource's service time expires while an application that leased it is using it, it leads to service failures.

3.1.2 Centrality Measures :

In graph theory and network analysis, indicators of centrality identify the most important vertices within a graph. They are scalar values given to each node in the graph to quantify its importance based on some assumptions. Centrality concepts were first developed in social network analysis, and many of the terms used to measure centrality reflect their sociological origin. There are a vast number of centrality measures used by analysts. In our approach, we use - eigenvector and closeness centrality measures.

Closeness Centrality : In a connected graph, closeness centrality (or closeness) of a node is a measure of centrality in a network, calculated as the reciprocal of the sum of the length of the shortest paths between the node and all other nodes in the graph. Thus, the more central a node is, the closer it is to all other nodes. It was introduced by A. Bavelas[7].

Eigenvector Centrality : In graph theory, eigenvector centrality (also called eigencentrality or prestige score) is a measure of the influence of a node in a network. Relative scores are assigned to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. A high eigenvector score denotes that a node is connected to many nodes who themselves have high scores [8].

3.2 Jarvis Scoring

The proposed method initially considers the list of all the components involved in a given cloud architecture that could potentially become faulty and list of all possible faults that arise due to individual components. One to many mapping is created between components and faults associated with the components. Any graph theoretic model relies on a precise establishment of relationships between nodes involved in the graph. In any given cloud environment, components in the environment can arise multiple faults. Subsequently, a fault could trigger a subset of other faults as we the entire cloud as one connected graph. Initially, faults represent nodes of the graph to be constructed. Every fault, initially, has an independent probability of arising regardless of being triggered by any other fault. These probabilistic weights are assigned to all faults which are mapped to cloud modules. These weights are based on the nature of the module and other external factors.

Subsequently, we define unweighted directed graph FG(V, E) whose V represent different faults associated to components and E represent whether or not will occurrence of a fault would trigger another fault. Impact factor relationship -¿ This tells how much of an impact does a fault has on another fault. One

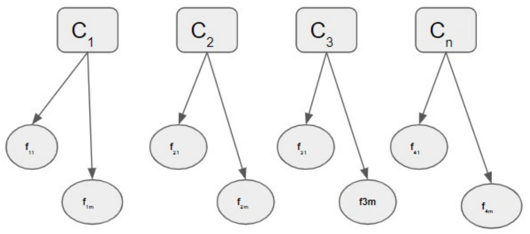

Figure 3.1 A graphical representation of different modules in a distributed computing model - C_1, C_2, C_3, C_n where multiple faults - f_{11} f_{12}, f_{21} f_{22}, etc. are caused by the failure or breakdown of modules C_1 to C_n

to One relationship. This depends on the independent probability of a fault's occurrence. A normalization constant is added to all the nodes that are involved in a cycle to stop the iteration when updating the node weights. It is experimentally seen that the efficiency and the accuracy of the node finding algorithm depends on the accuracy of probabilistic assumption of each node. With this graph established, we introduce a factor - impact factor - IF(f1, f2), representing the degree to which, the occurrence of a particular fault, trigger another fault. IF(f1, f2) is denoted as the weight of edge from node f1 to f2.

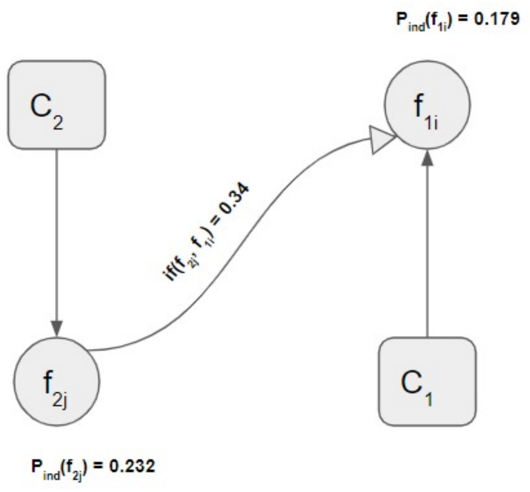
$P_{ind}(f_{1i}) = 0.179$
C_2
f_{1i}
$if(f_{2j}, f_{1i}) = 0.34$
f_{2j}
C_1
$P_{ind}(f_{2j}) = 0.232$

Figure 3.2 A small schematic graph consisting of two distributed components C_1 and C_2 where f_{1i} and f_{2j} are the faults caused by them respectively where if(f_{1i}, f_{2j}) is the influence fault node f_{1i} has on f_{2j}

Given independent probabilities of occurrence of all the faults in the graph, we iteratively update the probabilities after factoring it by the impact factor. For a connection between nodes f_1 and f_2, directed from f_1 to f_2, with P_1 and P_2 being initial independent probabilities of faults in nodes f_1 and f_2, and impact factor of f_1 on f_2 being IF(f1, f2) = $iff_{1,2}$, fault node weight is computed by

$$P_{2(\mathrm{i+1})} = P_{2(\mathrm{i})} * iff_{1,2} + \frac{\left[\sum_{n=1}^{k} P_{\mathrm{n}}\right]}{k}$$

where, we add the weighted average of all the contributing node weights involved in a cycle to terminate the iteration.

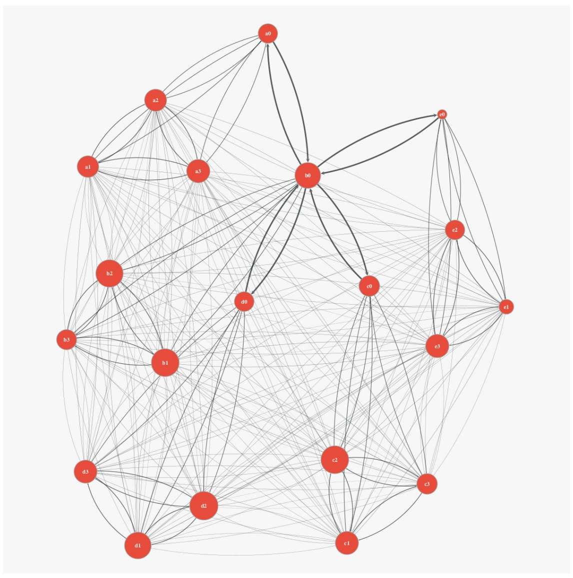

Figure 3.3 A graphical representation of subset of fault node graph with faults of different components connected with probabilistic edge weight denoted by networkx thickness

When any given cloud architecture is timely dynamic, we include the contribution of impact factor in computing the fault node weights at certain intervals for optimization and efficiency purposes. Otherwise, the importance of the fault nodes indicating the likelihood of its occurrence is computed considering the impact factor of other nodes directing to that particular node. With the node weights computed, closeness centrality measure and eigenvector centrality measures of every node, representing faults, is computed. Further, node weights, representing probabilities of fault triggers are normalized with the centrality measure values of each fault node in the range [0, 1].

$$EVR(f_\mathrm{n}) = \sum_{f_\mathrm{j} \in M(f_\mathrm{n})} \frac{EVR(f_\mathrm{j})}{L(f_\mathrm{j})} * iff(f_\mathrm{j}, f_\mathrm{n})$$

where EVR(f_n) is the eigenvector based rank for a fault node (f_n), and L(f_j) is the number of nodes that are connected to f_j and,

$$CR(f_\mathrm{n}) = \sum_{w \in G} \frac{1}{d(f_\mathrm{n}, w)} * iff(f_\mathrm{j}, f_\mathrm{n})$$

where CR(f_n) is the closeness centrality based rank for a fault node f_n and w is any fault in connection with (f_n) and d(f_n, w) is the shortest distance between f_n and w.

With the important nodes, representing the faults of high importance and high likelihood of occurring computed, the faults can be mapped back to the components that raise them to get an insight of whose replicas are needed the most and vulnerability rankings of cloud modules to perform subsequent localization algorithm on it.

4 EXPERIMENTAL RESULTS

We conducted tests with a total of two virtual machine instances simulated and over one hundred different faults that are related to close to twenty five modules from the host and hypervisor. With the module - fault mapping established, initial probabilities of fault occurrence for independent faults were taken from historical datasets[9]. Further, impact factor mapping between fault nodes were assigned based on frequency of dependency of one fault triggering module over another. With these values computed, the fault graph with probabilistic node weights is passed to two procedures to compute importance based on centrality measures based on both eigenvector centrality measure and closeness centrality measure. We simulated independent processes queued in, to receive the recent task information from the guest virtual machine instance. After dispatching operation, every task is sent to both the virtual machine instances by duplication methods. Frequency of dependencies of one node over another node and further, one particular fault related to a module over another fault of a different module can be generated.

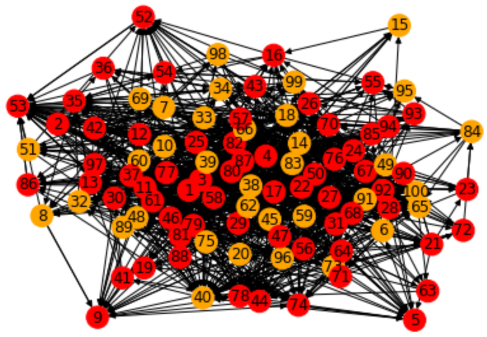

Figure 4.1. Graphical representation of fault node graph using networkx with nodes with eigenvector based centrality values greater than 0.08

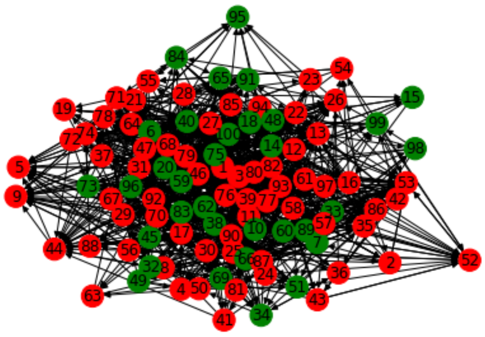

Figure 4.2. Graphical representation of fault node graph using networkx with nodes with closeness based centrality values greater than 0.08

A threshold value of 0.08 was set for ranks computed with the centrality based ranks and fault node weights to identify faults that are highly likely to occur to subsequently identify vulnerable components in the entire distributed system considered.

5 FUTURE WORK

The cloud paradigm has become a reality and has been adopted by researchers, IT industry and other organizations. Fault tolerance approaches are required to improve the quality of service in the cloud environment. However, the existing or proposed approaches have considered, in general, only the reliability issue of the simple cloud workflows and evaluation has been done using some basic metrics, such as response time, availability, throughput and reliability. Many studies model the system behavior including execution paths and component interactions. When the volume of data is enlarged, machine learning techniques are not appropriate in terms of the performance, so in such case, Deep Learning has been found to provide better performance in terms of accuracy [10]. Deep Learning uses various layers (hidden layer) of the nonlinear processing elements for the purpose of feature extraction as well as transformation. It is evident that deep learning approaches can also be integrated with the fault tolerance methods to predict the faults and take the corresponding preventive measures. Data De-duplication is an approach being increasingly used in cloud computing systems to reduce the data storage costs. Data de-duplication method is employed for removing redundant copies of data in cloud repositories in order to the decrease the storage space, upload bandwidth and consistency of data between multiple copies. Implementation of fault tolerance in systems employing data de-duplication can be challenging. Development of solutions that meet the reliability expectations while also decreasing storage costs and maintaining data consistency is a research topic where a lot of research work is active.

6 CONCLUSION

In this project we have implemented JARVIS, a centrality measure based fault localization technique to locate faults in distributed systems in a probabilistic manner. In our experiments, we conducted to test the suggested method with random generation of tasks in a simulated distributed computing environment simulated and injected faults. The results of these experiments with multiple centrality measures were analysed. We observed that this method based on probabilistic establishment of relationship between fault nodes is a practical and reliable way to rightly prioritize vulnerable cloud endpoints and modules.

References

[1] E. Y. H. E. Mohammad H. Alshayeji, Mohammad Al-Rousan, “”A Study on Fault Tolerance Mechanisms in Cloud Computing”,” *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, vol. 315, 09 2018.

[2] S. K. M. S, “”Graph Energy Based Centrality Measure to Identify Influential Nodes in Social Networks ”,” *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, vol. 315, 03 2019.

[3] S. M. A. A. S. M. N. E. E. Hemayed, “”Fault tolerance in cloud computing - survey”,” *2015 11th International Computer Engineering Conference (ICENCO)*, vol. 315, 02 2016.

[4] D. M. B. K. Ghamdan Mohammed Qasem, “”PROACTIVE FAULT TOLERANCE IN CLOUD DATA CENTERS FOR PERFORMANCE EFFICIENCY”,” *International Journal of Pure and Applied Mathematics*, vol. 315, 12 2017.

[5] E. A. M. E. A. El-Sisi, “”A reactive fault tolerance approach for cloud computing”,” *2017 13th International Computer Engineering Conference (ICENCO)*, vol. 315, 02 2018.

[6] A. S. S. M. Steinder, “”A survey of fault localization techniques in computer networks”,” *Scientific Data*, vol. 315, 06 2004.

[7] T. P.-R. F. W. Edith Cohen, Daniel Delling, “”Computing Classic Closeness Centrality, at Scale”,” vol. 315, 08 2014.

[8] A. B. M. K. Pandia, “”Eigenvector centrality and its application in research professionals’ relationship network”,” *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, vol. 315, 02 2015.

[9] L. M. C. M. M. P. O. R. R. Xin, “”Localizing Faults in Cloud Systems”,” *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, vol. 315, 05 2018.

[10] Y. Z. Weipeng Gao, “”A Cloud Computing Fault Detection Method Based on Deep Learning”,” *Journal of Computer and Communications*, vol. 315, 10 2017.