RELIABLE SOFTWARE SERVICES IN MULTI-CLOUD SYSTEMS

M.ALRIFAI[1] AND M. KHEMAKHEM[2]

Abstract:—Cloud computing is an emerging and innovative platform, which makes computing available to the end-users as services. . Most of the systems moved to the cloud for highly reliable service. Fault tolerance is very important to provide correct results even in the presence of a fault. Cloud providers offering product or service cannot easily transition to their competitors where customers become locked in; Multi-Cloud model invented to unify and combine many different clouds to allow software service portability, customer lock-in increases So, to achieve reliability in cloud services, the requirement for fault tolerance increases. In this paper, we introduce fault tolerance manager and select the most reliable provider and vote between an even or odd number of results to make decisions. It shows a good performance in executing complex tasks submitted and high resource utilization, and make decisions faster.

Index Terms—Cloud Computing, Fault Tolerance, Software reliability, Software Services, multi-cloud, Customer lock-in.

1. INTRODUCTION

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. The deployment is faster, cheaper and promotes on-demand, intelligent spending.

Cloud computing futures have caught the interest of different parties such as customers, companies, and governments to start using this model. Customers were once locked in with a single cloud provider platform. Then the multi-cloud model started to unify and combine many different clouds. The multi-cloud platform enhances the portability of software service between cloud service providers. The reliability of using single cloud service can have a direct impact on customer service reliability. There is a need to increase software service reliability by using many cloud providers running portable versions of the same software service.

There are two types of downtime for a computer system: planned and unplanned. Most of the time, planned downtime is kept to a minimum because providers are looking for high uptime in their services. So, planned downtimes are scheduled for computer resource maintenance or upgrades. Unplanned downtime arises suddenly due to human error, disaster or unexpected failure [1].

This paper aims to present a fault tolerance technique for improving the reliability of software services in the cloud. This technique is based on the integration of similar cloud services by multi-cloud systems for execution by a reliable cloud service provider.

Choosing to work with a multi-cloud is due to several factors. First, there is a benefit to the enhanced technology of cloud services [2]. Second, there is an increasing number of cloud providers that are offering products or services that cannot easily transition to a competitor [3], A U.S. report shows the top priority in selecting a solution provider in 2015 is the ability to personalize solutions. This leads to more lock-in problems. Third, as many as one-third of U.S. respondents have migrated a big percentage of their IT infrastructure to the cloud because of the need to improve reliability, reduce downtime, and offer improved service levels [4].

The goal of cloud interoperability and portability is to allow customers to make the best use of multiple cloud services that can interoperate and cooperate with each other, avoiding vendor lock-in. Cloud applications must be portable on top of many cloud PaaS and IaaS providers. This portability allows the migration between different providers to take advantage of lower prices and/or better qualities of services (QoS) [5]. [It is our concern to solve the fault in the cloud by using redundancy on a multi-clouds environment; therefore, our application needs to be portable.] Cloud portability regards the ability to transfer a cloud entity from one cloud to another; it has two main aspects: data portability and application portability[6][7][8].

2. METHODOLOGY

The purpose of the approach is to improve cloud application reliability using multi-cloud services.

Reliability can be improved by minimizing the effects of failure in cloud applications. The manager selects the available resource and uses a fault tolerance mechanism to generate multiple copies of the same application to run in multiple providers. Also, this approach is employed to build a complete application image to run in multi-cloud providers; this image is portable across the platforms and should be registered first in the cloud. [The majority voter is a dynamic approach which depends on the number of first received results.] Finally, the proposed approach needs to have a measuring mechanism and reliability assessor (RA) to evaluate cloud provider performance.

Handling fault by N-version programming (NVP) is a technique to tolerate software faults by creating a diverted duplication of the software. NVP is known as static redundant approach [9]. When software versions are independent, even if there is a fault that causes a local failure in version x, the whole system is likely to function correctly because the other independent versions are likely to function correctly under the control of the same fault-tolerance manager. We can ensure the design diversity by using different cloud service providers that have an independent design.

OS-level virtualization (OSLV) (software container) abstracts the operating system by isolating the applications; this will ensure by using containers to support portability across many cloud environments.

The fault-tolerance managerin Multi-Cloud (FTMMC) architecture is used to manage multiple, diverted, independent services to ensure reliability. The services are selected based on their rollability record and dispatch request in parallel, using NVP techniques. Based on the analysis of the results optioned by the fault-tolerance manager, we will update and improve performance records about cloud providers. This information will be used every time we run the cloud application to achieve optimal cloud application performance and reliability. The approach phases are described in the Figure 1.

The model based on best practices in fault tolerance in cloud computing [10][11][12],It consists of six phases: receiving a computing request, selecting ready, reliable providers, dispatching requests to a ready Cloud Service Provider (CSP), starting to watch time, gathering enough results to make a decision, and assessing the reliability.

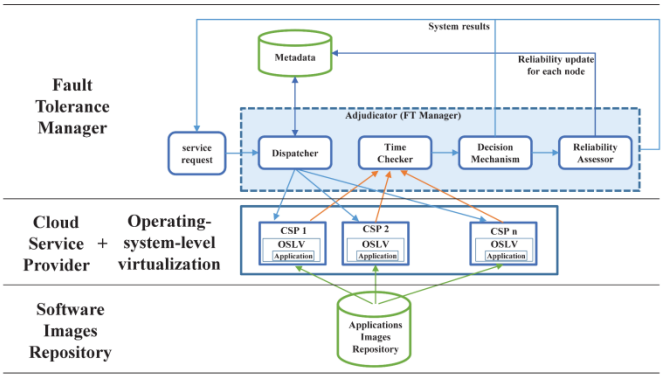

Fig1. FTMMC System Model

2.1 Select Ready Reliable Providers

The dispatcher selects the CSP for computation of the maximum reliability. The request has QoS requirements; these requirements play a role in the selection of, for example, criticality, duration, interval time checks (TC), and instances of speeding up. This will impact the number of nodes running and the type of selected resources.

2.2 Dispatching Requests to Ready CSP

The dispatcher component is used to allocate efficiently required resources and avoid under-provisioning and over-provisioning during failures. It also provides replication mechanisms by managing individual replicas of clients' applications, which include their location and current state. In order to connect with each provider, the set of metadata is required to establish a parallel connection with CSP and the OSLV inside it. Communication with each cloud provider service will use two methods. The first uses API functions to control the application remotely, and the second uses web service requests to distribute computing instructions to the cloud application. It starts once a new service request is received. Then, the number of versions is determined and selected. After that, the dispatcher passes the requests simultaneously to the nodes to be processed. A log record is done in order to keep information about the time of each stage and information.

2.3 Start Watching Time

Using a time checker will guarantee that the system will not have either late timing failure or early timing failure. The time checker is evaluated in milliseconds. The time checker starts after dispatching the requests to the nodes and keeps checking the status of the unfinished nodes. Node status updated in case of time

failure and exits the system when node status exceeds the time.

2.4 Gathering Enough Results to Make Decision

The steps are the core of fault tolerance manager because most of the fault tolerance process is covered here, as Decision Mechanism (DM) algorithm shows in Figure 2, starting with receiving a result from one of the nodes, the result is checked for early time failure. The decision can be made if two or more results are available; these results must have no time failures.

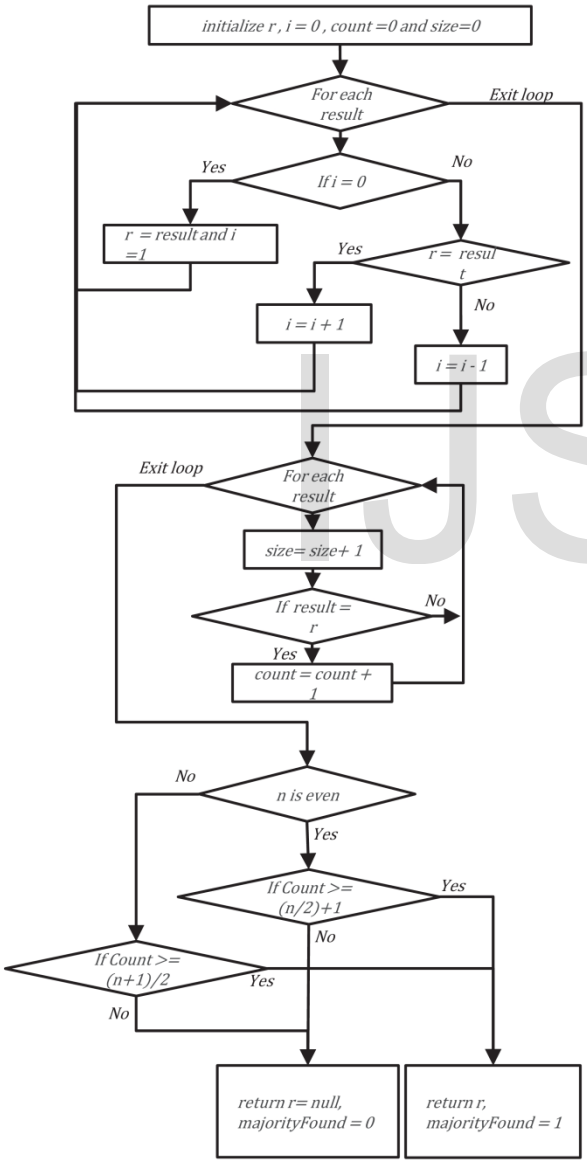

Fig.2 DM algorithm chart

An enhanced majority voting decision mechanism is being used, there are two enhancements done in the Moore majority vote algorithm, The first enhancement consists of checking if the algorithm has a majority result because the Moore algorithm has no indicator if a majority is not found; therefore, the counting loop is done after the Moore algorithm gives the results. The second enhancement is related to the majority rule, which can handle voting with an even or odd number of votes.

The goal of majority voting is to find the result that receives more than 50% of the votes. So, we use Equation (1) for even number of votes or Equation (2) for odd number of votes to find the minimum number of votes to reach a 50% majority [13].

$$\text{Minimum number of votes to majority} = n/2+1 \quad (1)$$

$$\text{Minimum number of votes to majority} = (n+1)/2 \quad (2)$$

After the decision mechanism success, the timer will be stopped, and the result will be submitted; otherwise, the system will wait for additional results to re-process the decision mechanism.

2.5 Assessing the Reliability:

In the case of failure or success in finding the correct result, the knowledge collected during the computing cycle can enhance the reliability rate of each CSP. In the beginning, the reliability of each CSP is supposed to be 100%. Later, if the node fails to produce the correct result or has a timely failure, its reliability decreases using adaptability factor n, and if a node produces a correct result without timely failure, its reliability increases using a reliability factor.

3. EXPERIMENTS and DISCUSSION

The performance of the approach was assessed with a different number of providers and different fault rates. Also, be comparing the performance with and without some of the components, finally, the approach was compared against the performance of a single-version software approach.

3.1 Throughput

Throughput is one of the most important standard metrics used to measure the performance of fault tolerance. It is used to measure the ability of the provider to accommodate service requests [14].

Throughput (n) = n/Tn, where n is the total number of submitted requests, and Tn is the total amount of time required to complete n requests.

The objective of this experiment is to compare the performance of using more providers in the manager. Also another objective is compare the performance of the

proposed dynamic majority voter for different number of received results and the majority voter for fixed number of received results.

In the first method DM receive the fastest two results, this two results processed by the DM to find if it good content between even number of results, if good content is not found the DM wait for the next result, if a third result received the DM process this results to find good content between an odd number of results , if good content is not found the DM wait for more even and odd number of results, if the maximum number of results reached the manager send all results to RA , finally, the manager start new cycle for the same request or for the next request.

In the second method DM wait only for all results to be received, the results processed by the DM to find if it have good content, next, the manager send all results to RA, finally, the manager start new cycle for the same request or for the next request.

In both methods, 2500 requests were submitted, the experiment start with an average of 5% injected faults and finishing with 35% injected faults.As Figure 3 and 4 shows.

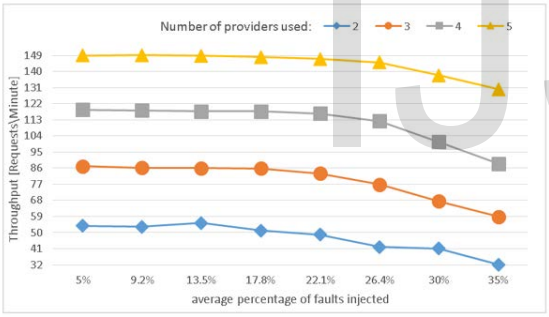

Fig.3 Throughput Comparison with 2500 requests submitted and voting start with two or more results received

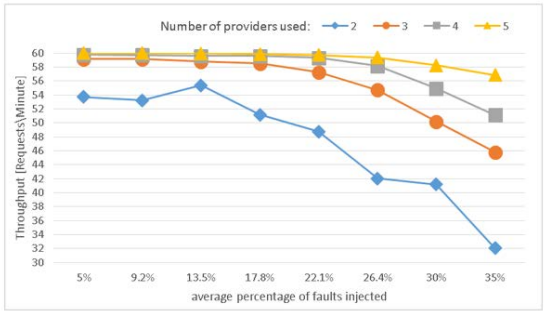

Fig.4 Throughput Comparison with 2500 requests submitted and voting start only when all results received

The approach can tolerate the fault very well, the approach can benefit from the increase number of cloud to finish more requests. Also, the approach can benefit from voting using dynamic number of results to speed up the process of DM.

The number of provider had the higher positive effect in the first method performance then second method, add more provider in the first method have no limit by the approach to give high performance and the reliability.

3.2 Turnaround time (TRT):

The turnaround time is an important parameter for evaluating the performance of fault tolerance. It shows how long the system takes to execute customer requests; a higher number of TRT time means lower productivity [14]. TRT is the interval from the time of submission requests to the time of completion.

This experiment compares the performance of multi-cloud fault-tolerance technique with single-cloud fault-tolerance technique against different injected faults.

In the first method, a multi-cloud use many provider to receive many results for the same request, request executed in parallel by many providers, TC check the time and result readiness in parallel with provider execution, next the DM and RA work in sequence to process the results.

In the second method, a single-cloud use single provider to receive single result, this result checked for fault using Reversal Checkers acceptance test, it work by reversing the computing using the output value to result the corresponding input value [15].

In this experiment, 2,500 requests are submitted, 32 cases were tested by starting with an average of 5% injected faults until 35% injected faults. As Figure 5 shows.

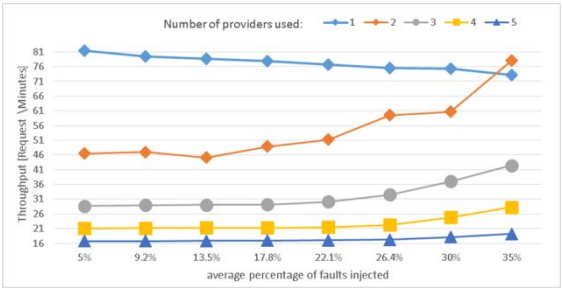

Fig.5 Turnaround time comparison with 2500 requests submitted

Reversal checkers in a single cloud will lead to more delay time to accept the result, this result high TRT time. On the other hand, the proposed approach selects the best providers who are less likely to fail. This will lead to

fewer faulty providers and reduce the delay times compared to the other approach.

Using two providers in the proposed approach result high TRT compared to use higher number of used providers. Using only two provider can give same result as single-version approach if the injected failure exceed around 33%.

4. CONCLUSION

In this paper, we have built a multi-cloud fault tolerant to enhance the reliability in the cloud. Our approach implements fault tolerance technique to assess the reliability and vote faster by making a decision for both odd and even number of results.

We have performed a series of experiments to assess the performance of the proposed multi-cloud fault tolerant approach in many situations. Results show that the use of our fault-tolerant approach reduces the wasted time caused by faults to near optimal level system throughput or very low turnaround time.

REFERENCE

[1] Jeenia Jain, and Ramandeep Singh, "Improving Service Reliability in Cloud Computing Environment," *International Journal of Scientific & Engineering Research,* vol. 5, no. 3, pp. 2229-5518, March-2014.

[2] T. McCue, "Cloud Computing: United States Businesses Will Spend $13 Billion On It," forbes.com, 29 jan 2014. [Online]. Available: https://www.forbes.com/sites/tjmccue/2014/01/29/cloud-computing-united-states-businesses-will-spend-13-billion-on-it/#788114734f89. [Accessed 12 jan 2017].

[3] A. Rabbetts, "Cloud supplier lock-in – our experience," computerweekly, Dec 2013. [Online]. Available: http://www.computerweekly.com/opinion/Cloud-vendor-lock-in-our-experience. [Accessed Jan 2017].

[4] C. Kern, "Survey: Reliability, Business Continuity Drive Demand For Cloud Adoption," Business Solutions Magazine, Jan 2015. [Online]. Available: https://www.bsminfo.com/doc/survey-reliability-business-continuity-drive-demand-for-cloud-adoption-0001. [Accessed jan 2017].

[5] Swaraj P. Thakre, and Prof. Nitin R. Chopde, "A Review of Collaboration of Multi-Cloud – An Effective Use of Cloud Computing," *International Journal of Application or Innovation in Engineering & Management (IJAIEM),* vol. 2, no. 3, p. 328, March 2013.

[6] C. S. C. Council, "Interoperability and Portability for Cloud Computing: A Guide," Nov 2014. [Online]. Available: http://www.cloud-council.org/deliverables/CSCC-Interoperability-and-Portability-for-Cloud-Computing-A-Guide.pdf. [Accessed 2 Mar 2016].

[7] Tobias Kurze, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai, and Marcel Kunze, "Cloud Federation," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011),* Rome, Italy, sep 2011.

[8] T. o. Group, "Cloud Computing Portability and Interoperability," 13 4 2016. [Online]. Available: http://www.opengroup.org/cloud/cloud_iop/. [Accessed 3 Feb 2016].

[9] V. M. Sivagami, and K. S. EaswaraKumar, "Survey on Fault Tolerance Techniques in Cloud Computing Environment," *International Journal of Scientific Engineering and Applied Science,* vol. 1, no. 9, pp. 419-425, 2015.

[10] Anjali D.Meshram,A.S.Sambare, and S.D.Zade, "Fault Tolerance Model for Reliable Cloud Computing," *International Journal on Recent and Innovation Trends in Computing and Communication ISSN 2321 – 8169,* vol. 1, no. 7, pp. 600-603, 2013.

[11] N.Chandrakala,P.Sivaprakasam, "Reliable VM Identification in Multi Cloud Environment," *International Journal of Computer Applications,* vol. 65, no. 15, pp. 8-11, Mar 2013.

[12] Rajesh.S, and Kanniga Devi.R, "Improving Fault Tolerance in Virtual Machine Based Cloud Infrastructure," *International Journal of Innovative Research in Science, Engineering and Technology,* vol. 3, no. 3, pp. 2163 - 2168, Mar 2014.

[13] K. J. Smith, Nature of Mathematics Thirteenth Edition, Boston, MA USA: Cengage Learining, 2015.

[14] Fiaz Gul Khan, Kalim Qureshi, and Babar Nazir, "Performance evaluation of fault tolerance techniques in grid computing system," *ELSEVIER Computers and Electrical Engineering Journal,* vol. 36, no. 6, p. 1110–1122, Nov 2010.

[15] Dubrova, Elena, Fault-Tolerant Design, springer, 2013.

[16] Docker, "Docker Overview," [Online]. Available:

https://docs.docker.com/engine/understanding-docker/. [Accessed 11 Nov 2016].