ISSN: 0975-766X
CODEN: IJPTFI
Research Article

Available Online through
www.ijptonline.com

FAULT TOLERANCE IN CLOUD ENVIRONMENT BASED ON HIGH-LEVEL COMPUTATION MANAGEMENT

Senthilnathan Palaniappan[1] *, Biswa Bhusan Ghosh[2] and Dr. Arun Kumar Sangaiah[3]
[1]School of Computing Science & Engineering, VIT University, Vellore, India.
[2]School of Computing Science & Engineering, VIT University, Vellore, India.
[3]School of Computing Science & Engineering, VIT University, Vellore, India.
Email: senthil.me.psg.it@gmail.com

Received on 04-08-2016 *Accepted on 03-09-2016*

Abstract

As increasing in the perspective of cloud computing, the more number of customer get into involved to access through it. So for the organization it’s very much important to take care of the customer in order for Quality of Service (QoS). For QoS it has to consider about on the availability and reliability of the system. In this paper, we proposed a system-level approach to solve the above problem along with reduce the complexity of the system and making a good communication between the customers which will increases the profitability for organization. Here for maintaining client requirement based on fault tolerance level of service used one hybrid migration algorithm which balanced the multiple hosts that are being present in the clustered storage, client’s application deployed into the infrastructure storage i.e. in data center being cloned into different instance of VMs in node. The infrastructure provider barely communicates with customer feedback and fault support as well, which lead the system on high level.

Keywords: Cloud computing, fault tolerance as service, system level fault tolerance, hybrid algorithm.

1. Introduction

The increasing in the demand of acquiring what’s more, discharging registering resources in a financially way has brought about a wide application in cloud computing worldview. The accessibility of an extensible pool of resources for the client gives a compelling distinct option for convey applications with high adaptability and handling prerequisites[1-3]. By and large, a Cloud registering foundation is worked by interconnecting expansive scale virtualized server farms, furthermore, registering resources are conveyed to the client over the Web as an on-interest administration by utilizing virtual machines[1-6]. While the advantages are tremendous, this figuring worldview has fundamentally changed the measurement of dangers on client's applications, particularly in light of the fact that the failures (e.g., server overload, network congestion, hardware faults) that show in the server farms are outside the extent of the client's association. In any case, these

failures force high ramifications on the applications sent in virtual machines and, therefore, there is an expanding [7, 8] need to address clients' reliability and availability concerns.

Distributed computing environment, it's extremely hard to plan a superior adaptation to internal failure arrangement that productively consolidates the both disappointment conduct and framework engineering of utilization. The trouble emerges because of the high multifaceted nature of framework i.e. the framework reaction time and for[9-12] steady layer of cloud it doesn't give the entire measure of data to its client.

Virtualization innovation has pulled in extensive enthusiasm for the distributed computing in late years. Server farms total a wide range of resources (e.g. information, programming, equipment) to give different administrations virtualization innovation. With a specific end goal to enhance the execution of the server farm by [3,5,13] augmenting the throughput and minimizing the reaction time of the framework, it is important to adjust loads among the physical hosts for the cloud environment. As appeared in (Figure1), in the cloud computing environment, the server farm has numerous VMs to run the administration, and the fault tolerant level of the administration is ensured by appropriating the VMs of the administration onto different physical hosts[14,15]. Fault tolerant level can be portrayed as: if administration i can work ordinarily when k_i has separate, the fault tolerant level of administration i is characterized as k_i. To give dependable administrations, the fault tolerant level ought to be guaranteed while moving VMs to adjust loads. As outlined in (Figure 2 and 3), two VMs are moved from hosts 1 and 2 to hosts 3 and 4, separately, to adjust loads. In any case, the adaptation to internal fault level of administration VM2 is diminished while the load is adjusted. In the second case, where the customer newly sending request to the provider[1-20]. The organization start generating a space for that user but doesn't mean it would satisfy for multiple number of customer. So here we have been proposed a hybrid algorithm where it has done automatically allocate without violating the fault-tolerant level of service[2]. Where the scenario like in the (Figure4) shown below.

In this paper, it has been focused more on the customer side instead of doing everything by the provider side. In real life scenario we have seen most of example among them banking service is very good example so far as per our proposed work[2,20,6]. As an example, it has considered that client who is allowed to their own service instead of banker's presence or in holiday. This is the case where client can able to manage their account, online transaction, and net banking through the Internet. Here in this scenario, the provider previously has made it's architecture such that whenever the fault occur, it understand in fraction of seconds and the server goes slow or down then make them availability[21-29].

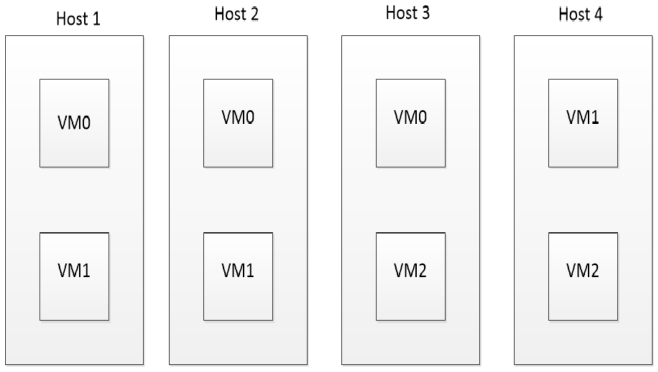

Figure 1. Several VMs running the same service.

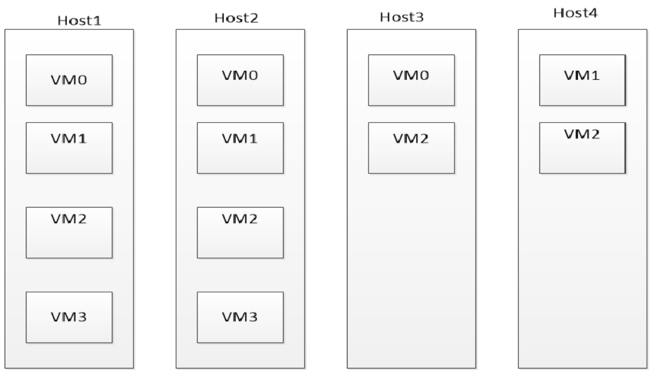

Figure 2. Load unbalancing.

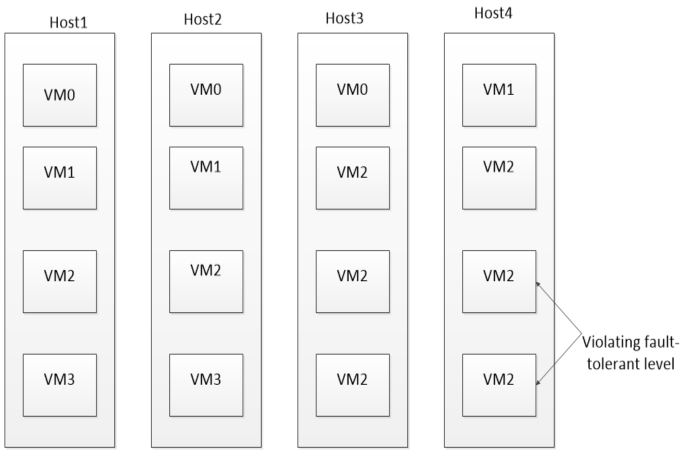

Figure 3. Violating fault-tolerant level service.

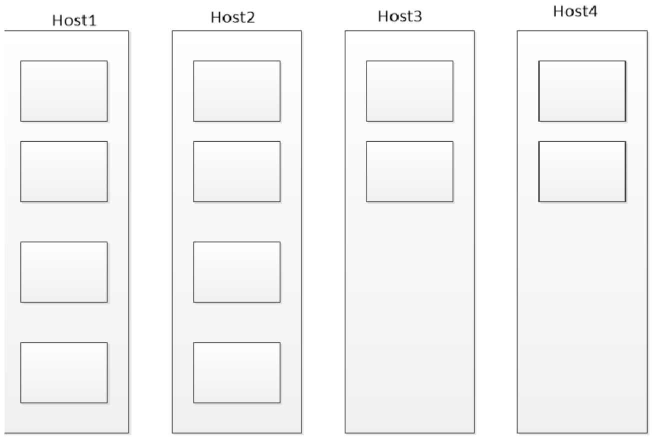

Figure 4. Not allocated service & unbalanced.

The rest of this paper, organized as following way. The section 2, described related works. In the section 3, proposed works & algorithm. In the section 5, simulation result. In section 6, conclusion & future work.

2. Related Works

In traditional methodology, it has pushed another estimation where applications sent in a Cloud handling infrastructure can get required fault tolerance properties from a pariah. To reinforce the new estimation, we enhance our work and propose an approach to manage recognize general fault tolerance instruments as self-governing modules such that each module can direct work on customers' applications. We then enhance each module with a game plan of metadata that portray its fault tolerance properties, and use the metadata to pick instruments that satisfy customer's requirements. Also, it showed an arrangement that: 1) passes on a complete fault tolerance answer for customer's applications by joining picked fault tolerance frameworks, and 2) discovers the properties of a fault tolerance course of action by technique for runtime watching. Considering the proposed approach, we plot a structure that successfully facilitates with the present Cloud infrastructure and empowers a pariah in offering fault tolerance as an organization[1]. Scientific workflows can profit by SIs with a viable bidding and a productive fault tolerant system. Such a system can endure out-of-bid failures and lessen the expense massively.

To ensure the fault-tolerant level of all administrations gave by the server farm while adjusting the heap taking into account VM movement among the hosts, a novel burden adjusting plan, Guarantee fault-tolerant necessity load Balancing Based Solution(GFTLBS) in view of VM relocation, is proposed in this paper[3]. This paper points to give a superior comprehension of fault tolerance challenges and recognizes different instruments and strategies utilized for fault tolerance. At the point when different examples of an application are running on a few virtual machines and one of the server goes down, there is a need to execute an autonomic fault tolerance procedure that can deal with these sorts of faults. To address this issue, cloud virtualized framework engineering has been proposed and actualized utilizing HAProxy. The proposed engineering additionally has been approved through exploratory results.

Fault tolerance is fundamental for the framework to ensure both accessibility and dependability of basic administrations and application execution. To minimize the event of failures in the framework and its effect on application execution, failures ought to be taken care of by appropriate system[15]. The review reason for existing fault tolerance strategies in cloud figuring is to think about the deficiencies of these current systems and creating new calculation for taking care of fault betterly. This paper talks about different parts of faults and the requirement for fault tolerance in cloud processing.

Wellbeing basic ongoing frameworks require working appropriately to evade failure, which can bring about money related misfortune and setbacks. So there is an expanded need to endure the fault for such kind of frameworks to be utilized with cloud infrastructure[12]. For this reason they had displayed a model for the fault tolerance of constant applications running at cloud infrastructure. Firstly, they requests information ability and experience from clients as far as choice, arrangement, and coordination of uses with accessible fault tolerance structures. Furthermore, the granularity at which applications can deal with failures, and resource costs acquired by the applications throughout the fall flat free period remains consistent. In conclusion, deliberation layers of the Cloud engineering result in non-straight forward and rigid situations requiring an excess of push to designers as meager data about the basic infrastructure is accessible[14,6].

In this paper, it has the primary endeavor to describe server failures for substantial server farms. They show a definite examination of failure attributes and investigate the relationship between the failures and a substantial number of variables, for example, time of the machine, the quantity of hard plates it has, and so on. This is the primary work to measure the relationship between progressive failures on the same machine. We find that the experimental information fits a reverse capacity with high criticalness. We perform the principal prescient investigation in a datacenter to mine for variables that clarify the explanation for failures. We find, for occurrence, that the datacenter where a server is found is an incredible marker of failures as is the producer. We indicate exactly that the unwavering quality of machines that have as of now seen an equipment failure in the past is totally not the same as those of servers that have not seen any such[7] occasion. Fault tolerance procedures presented a three surely understood systems are in the accompanying with recipes for ascertaining the failure probabilities of the fault tolerant modules. Our work will for the most part be headed toward the execution of the system to quantify the quality of fault tolerance benefit and to make a top to bottom investigation of the money saving advantages among every one of the partners. This paper considers three possibly practical strategies for burden adjusting in substantial scale Cloud frameworks[2]. Firstly, a nature-stirred estimation may be used for self organization, fulfilling overall weight changing by method for close-by server exercises. Moreover, self-affiliation can be incorporated bringing with record unpredictable testing of the system range, giving a balanced weight over all structure nodes. Thirdly the system can be revamped to enhance work task at the servers. This paper expects to give an appraisal and relative examination of these methodologies[3], demonstrating scattered computations for weight modifying.

In this paper, they considered three perhaps useful strategies for weight modifying in immense scale Cloud systems. Firstly, a nature persuaded estimation may be used for self organization, achieving overall weight modifying through adjacent server exercises. Besides, self-affiliation can be composed considering sporadic investigating of the system space, giving a balanced weight over all structure nodes. Thirdly the system can be remade to streamline work undertaking at the servers. This paper hopes to give an evaluation and close examination of these techniques, showing passed on computations for weight adjusting [20].

3. Proposed Work

In this paper, it has been proposed a hybrid algorithm which is combination of two different concept that discussed in below section. In section 3.2, described about the existing architecture. In section 4, hybrid algorithm that we proposed.

3.1 Basic Concepts

The mostly used concept to reduce the fault tolerant is based on the redundancy. In this concept, each component keeping its duplicate copy i.e. replication of each components. Hence, the copy of the component such as hardware, software, and network resources which helps in availability of the resources. For example, in the banking service each customer’s data replicate in different server (at least one copy of each) and that copy of customer keeps for its queries purpose. In active replication method all the redundant components are concurrently called, but which utilizes more resources than the passive method of replication where only single processing node can handle the request at time of backup. We watched that an administration supplier must fulfill the prerequisites of customer, through a viable acknowledgment of its usefulness in an association. The administration supplier must outline in a manner that it would coordinate with existed cloud infrastructure which depicted in segment 3.2.

3.2 System Architecture

In this segment, we display a theoretical system, the Adaptation to Fault Tolerance Manager (FTM), that gives the premise to an administration supplier to understand the conveyance plan introduced in the past segment and consequently to offer adaptation to non-critical failure as a administration. We plan to embed our structure as a faithfulness layer between the customer's applications and the equipment that works specifically on the highest point of the virtual machine administrator at the level of VM occasions. The Fault Tolerance Manager must address the issue of heterogeneity in processing resources, satisfy the objective of straightforwardly giving adaptation to internal failure backing to client's applications against node failures, and fulfill adaptability and interoperability objectives.

To defeat these challenges, we propose to assemble the Fault Tolerance Manager utilizing the standards of administration situated design, where each ft−unit is acknowledged as an individual web administration, and a ft−sol is shaped utilizing the Business Process Execution Language (BPEL) develops.

3.2.1 Client Interface

The administration process starts when a customer demands the administration supplier to offer adaptation to internal failure backing to its applications with a desired arrangement of properties. In this setting, it is crucial to incorporate a customer interface segment inside FTM that gives a determination dialect that permits customers to indicate what's more, characterize their prerequisites. Nevertheless, since the present-day appropriated registering structures require its customers to manage their VMs while overseeing cutting edge system level concerns, an automated game plan gadget that obliges clients to simply pick the application for which they wish to gain adjustment to non-basic failure support, and correspondingly give estimations of reliability, availability, response time, criticality of the application and cost can be useful. We take note of that a motorized setup instrument can compel human errors and sparing time by reducing the necessity for manual dull outline. Moreover, if the data can be given in an irregular state association, (for instance, rates, range, and numbers), undoubtedly, even customers with a nontechnical establishment can plan the fancied properties easily. We consider the point of view of changing strange state metric qualities into an arrangement of inadequacy flexibility properties, and translating the properties to the extent standard adjustment to non-basic failure instruments as a noteworthy part of our future work.

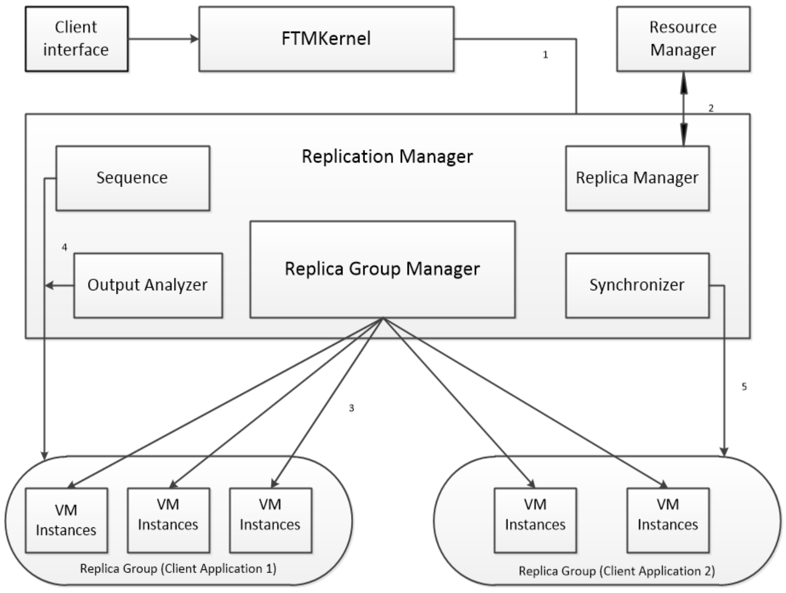

Figure 5. Diagram of different parts in replication manager, and their cooperation with different segments of the structure.

3.2.2 FTMKernel

The main part of our structure is the FTMKernel that is in charge of creating a deficiency resistance arrangement in view of customer's necessities utilizing the web administration modules (ft−units) actualized by the administration supplier, conveying the formed administration on customer's applications, what's more, checking every administration case to guarantee its QoS. FTMKernel is made out of an organization registry, a piece engine, and an appraisal unit. Despite the asset boss, client interface, and FTMKernel, we observe that our structure must fuse an arrangement of parts that give an equal sponsorship to accuse strength instruments. These sections basically impact the way of organization offered by the organization supplier, and are key to satisfy client's necessities and goals. We fuse the going with fragments in our framework, and present the general designing of the Fault Tolerance Manager.

3.2.3 Resource Manager

The administration supplier must keep up a predictable perspective of all registering resources in the Cloud to proficiently distribute resources during every customer ask for and to keep away from over provisioning during failures. In this connection, an resource chief that consistently screens the working condition of physical what's more, virtual resources keeps up a database of stock and log data, and a chart speaking to the topology and working condition of resources must be presented by the administration supplier in the foundation supplier's framework. The database of the resource administrator must keep up the stock data of every machine, for instance, its stand-out serial number, game plan of the machine (e.g., processor speed, number of hard disk, and memory modules), date when the machine was dispatched (or decommissioned), region of the machine in the gathering.

3.2.4 Replication Manager

This part underpins the replication system by calling replicas and managing their execution in perspective of the client's necessities. We mean the game plan of VM cases that are controlled by a single execution of a replication component (ft–unit) as a replica bundle. Each replica inside a social occasion can be uncommonly perceived, and a course of action of tenets that must be satisfied by a duplicate get-together are resolved. The endeavor of the replication head is to make the client see a duplicate pack as a solitary organization, and to ensure that the blemish free replicas demonstrate right lead amid execution time. (Figure5) gives an audit of various sections inside the replication executive and their associations with each other. To bolster a replication framework, the duplicate invoker first considers the pined for repli cation parameters, for instance, the style of replication (dynamic, idle, cold uninvolved, hot idle), number of replicas, and prerequisites on relative circumstance of solitary replicas, and structures the duplicate pack. At the end of the day, the replica invoker takes the reference of a client's application as information from FTMKernel, analyzes the typical adjustment to non-basic failure properties, and coordinates with the asset boss to gain the range of each replica. The repl ica pack director then makes the replica bundle by conjuring VM events at those regions and managing their execution. The sequencer gives the data to application executing in the replica bundle by strategy for assention tradition with a particular finished objective to ensure determinism among replicas. The yield analyzer finishes larger part voting on the responses got, besides, the picked result to the client. The synchronizer consolidates methodologies to overhaul the state of fortification replicas with that of the crucial in a duplicate group. It similarly bolsters interest change and vital race counts when the crucial hub encounters failure. We observe that energy of these approach, as it were, add to the consistency and relentless nature of the organization.

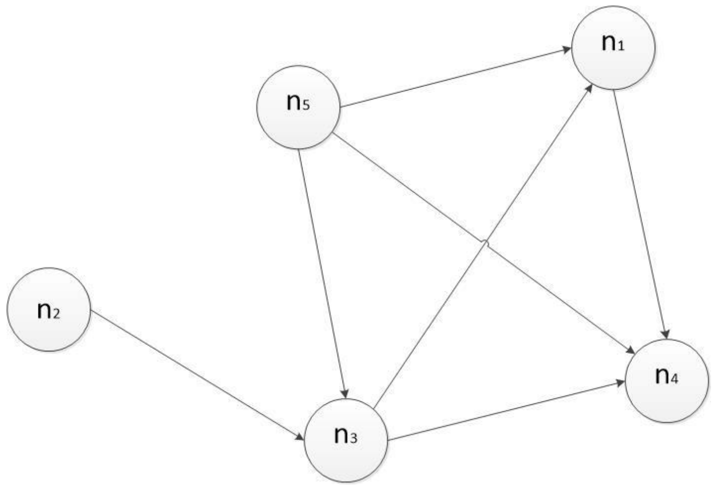

Figure 6. Occasion of resource diagram produced by resource manager.

Representation: Let us consider that FTMKernel picks a uninvolved replication instrument contrasting with the sparing cash organization's interest where the going with replicas must be satisfied: 1) the replica pack must contain one vital and two fortification hubs at all times; 2) the hub on which the crucial executes must not be conferred to whatever other VM samples; and 3) each one of the replicas must be arranged on different hubs. For the Cloud establishment depicted in (Figure6), the replication boss shapes a duplicate social event of the dealing with a record organization's application by picking the hub n1 for the crucial, and hubs n3 and n4, independently, for support duplicates. We observe that n3 and n4 can have VM instances of other duplicate bundles, while one and just VM event can continue running on n1. The synchronizer of replication boss once in a while checkpoints the vital and upgrades the state of support duplicates.

4. Hybrid Migration Algorithm

Requirement: The placement of VMs in the host in the data center, placement; the source host index denoted as s; the destination host index denoted d;

Ensure: The number of hosts that have been balanced in this procedure: one host or more than one, and empty i.e. more than one VMs will be free;

Initialization: Here two parameters considered as

Heaviest_load=0;

```
                Lowest_load=0;
                Srs_host=heaviest_load;
        Des_host=lowest_load;
```

Reallocation(s,d)

```
1. For i=0; i<n; i++ do
2.    Srs_host=placement[i][s];
3.    Des_host=placement[i][d];
4.    Difffrence_of_VMs_of_host[i][s]=Srs_host- Des_host;
5. End of fo
6. Sum(Srs_host);
7. Sum(Des_host);
8. For i=0; i<n; i++ do
9.   If (Difffrence_of_VMs_of_host[i][s]>1 && Difffrence_of_VMs_of_host[i][s]>2) then
10.  Reallocate the 2 instance from heaviest load to the lowest load;
11.  Difffrence_of_VMs_of_host[i][s] --;
12.    Else
13.  Go for single instance reallocation;
14.    End if
15.   End of for
16.  Return 0;
```

Allocation(s,d)

```
1. For i=0; i<n; i++ do
2.   Placement[i][s]=0;
3.   Placement[i][d]=0;
4.   Srs_host=placement[i][s];
5.   Des_host=placement[i][d];
6. End of for
7. while(placement[i][s]!=0) do
```

```
8.  Allocate the VMs instance based on clients     requirement randomly;
9.      Placement[i][s]++;
10. End of while
11.   For  i=0; i<n;i++ do
12.   If (host is not balanced) then
13.    Call reallocation(s,d);
14.   Else if(balanced)
15.   Return 0;
16.   End of if
17.  End of for
```

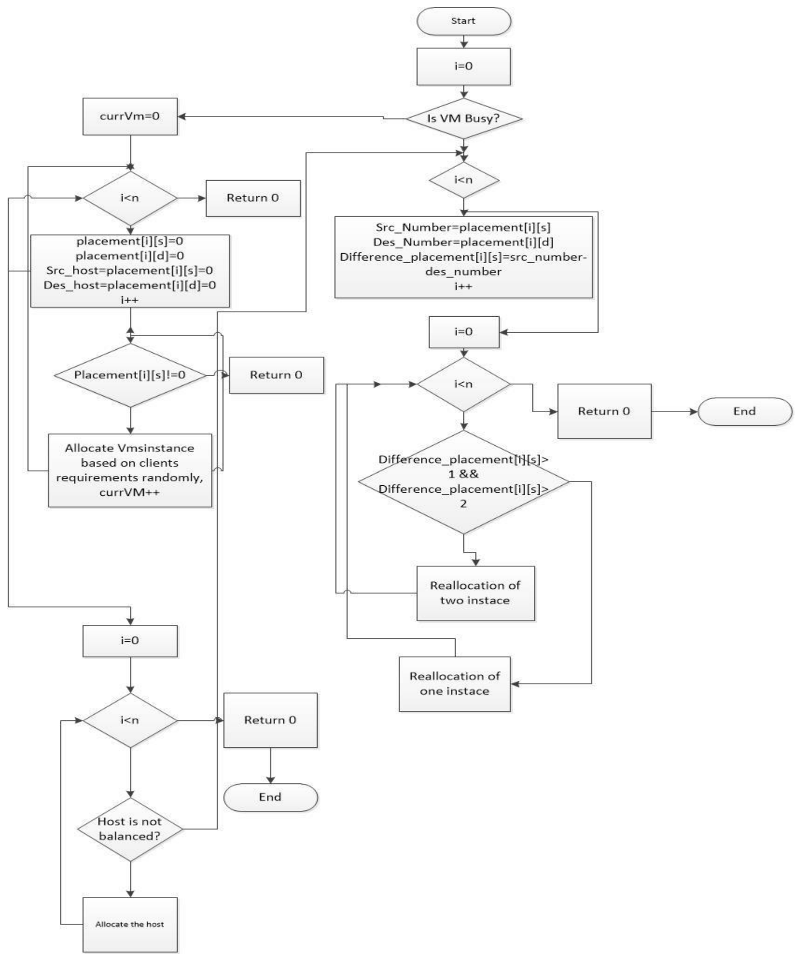
Start
i=0
Is VM Busy?
currVm=0
i<n
Return 0
placement[i][s]=0
placement[i][d]=0
Src_host=placement[i][s]=0
Des_host=placement[i][d]=0
i++
Placement[i][s]!=0
Return 0
Allocate Vmsinstance based on clients requirements randomly, currVM++
i=0
i<n
Return 0
End
Host is not balanced?
Allocate the host
i<n
Src_Number=placement[i][s]
Des_Number=placement[i][d]
Difference_placement[i][s]=src_number-des_number
i++
i=0
i<n
Return 0
End
Difference_placement[i][s]> 1 && Difference_placement[i][s]> 2
Reallocation of two instace
Reallocation of one instace

Figure 7. Flow chart of hybrid algorithm.

Movement calculation chooses some VMs from source host and moves the VMs to the destination host without disregarding the Fault tolerant necessities, where Difference_palcement[i][s] is utilized to hold the subtract number of the VMs of administration i running on the host s and host d. (Figure7) shows the stream outline of movement calculation. At the underlying step, the distinction of the number of all administrations on the source host and the destination host. At that point, if the distinction of particular administration is bigger than 1, at that point this administration will be relocated. On the off chance that loads of both the source host and the destination host are equivalent to the normal number of VMs, the calculation will return 2. On the off chance that there is one and only host (the source host or the destination have) whose loads are equivalent to the normal number of VMs, the calculation will return 0.

Here we have introduced one additional concept of hybrid algorithm which is combination of both reallocation and the allocation of virtual machines.

5. Simulation Result Discussion and Comparison

CloudSim is a reproduction programming of distributed computing, outlined and executed. CloudSim[18] has two new favorable circumstances: (1) demonstrating and recreation of vast scale distributed computing foundation; (2) an independent supporting Data Center, Data Center Broker, scheduling and allocation policy of system, and it utilizes virtualization to give resource. The resource of a host in a server farm, for example, physical host, could be mapped to a few VMs taking into account the client necessity, along these lines there are conceivable diverse quantities of VMs running on the hosts. Thusly, VM movement is important to adjust the load.

Keeping in mind the end goal to exhibit that can promise the fault tolerant level of all applications with load balancing in view of VM movement, the impact on the fault tolerant level is assessed by the reenactment taking into account CloudSim. Fault tolerant level is characterized as: if the administration can be typically given when N has separate, then the fault tolerant level is characterized as N.

The essential VM number for administration i is depicted as: if n VMs of the administration i can fulfill the execution necessity of administration i, while n-1 VMs can't fulfill, then the fundamental number of VMs of administration i is characterized n.

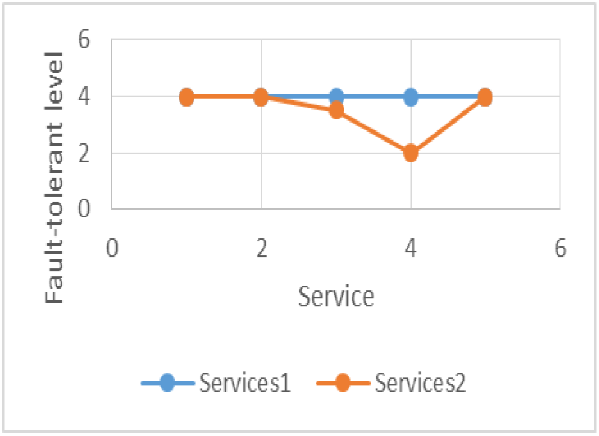

Figure 8. Fault-tolerant level of 5 hosts and 5 services.

The experiment consisted of several VMs running in different hosts of data center. All VMs were on one physical computer with a 4-core 8-thread 3.5GHz Intel core i3 CPU. Load was generated by customer a constant that could be handled by 1 VM. Load was balanced and the experiment started with all VMs & one was killed every 15 min. The same was then replicated CloudSim.

We set up one user base with 1000 user sending 32 req/h each. There was one data center with one host with 8 processors. Computational complexity was obtained from minimal measured latency, where the number of customer sending their request with appreciated delay to get response.

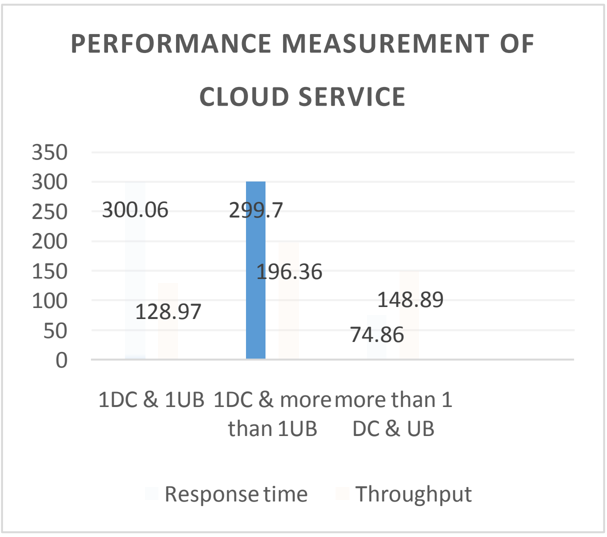

Figure 9. Performance of cloud Service.

In the above (Figure8) has been shown that the cloudSim[18] result of different request that has been sent by the different clients.

Here, it has found some cases including existing work and our proposed work. In existing work they have been proposed the individual customer how get response with better service from the cloud environment. Here our proposed work has maximized the throughput with the reducing the response time, which reduced the time complexity by making the system work pretty easier and faster way. In the (Figure9) above it shows that there is increasing the throughput by decreasing the response time.

With a specific end goal to Hybrid Algorithm Virtual Migration(HAVM) the adaptation to non-critical fault level of all the administrations in the load balancing process in view of VM relocation, distinctive gatherings of the quantity of VMs, hosts, essential VMs and adaptation to internal fault level of administrations are tried in the recreation. As we saw that here in the above graph there is result as compare to the Existing Virtual Migration (EVM)[3] is better in some parameters and some parameters are having limitation.

Here the main aim was that to reduce the time complexity i.e. the response time and maximize the throughput as a result we got.

The rest of comparisons are mentioned below in (Table 1). Here some parameters are having disadvantage over our proposed algorithm but it reduces the response time of the service by considering the fault level of cloud infrastructure.

Table 1. The comparisons of proposed algorithm and the existing algorithm.

Various Parameter	**Hybrid Algorithm Virtual Migration (HAVM)**	**Existing Virtual Migration(EVM)**
1. Response Time(in ms)	74.85	300.6
2. Cost Effective(in dollar)	2.5 of VM cost and .38 of data transfer cost	.5 of VM cost and .06 of data transfer cost
3. Throughput(in ms)	148.89	128.97
4. CPU Utilization	It took long time	It took less time
5. Latency	Delay between source to destination takes less time	Same
6. Data Transfer Time(In ms)	.065	.064
7. Min and Max Time(in ms)	Min-37.13 & Max-248.15	Min-237.059 & Max-369.115
8. Data Center Processing Time(in ms)	.44(average) Min-.02, Max-.90	.34(average) Min-.02, Max-.61
9. Data center Requesting time(in ms)	.464(average) Min-.021, Max-.886	.342(average) Min-.19, Max-.612

6. Conclusion and Future Work

In this paper, we proposed an approach for realizing fault tolerance along with we proposed a hybrid algorithm where it explained on above section. The proposed approach when combined with existing one Fault Tolerant Manager and hybrid one shows significant role in this paper. For this two role classified from the customer's satisfied of getting more benefit from service provider. Here it reduced the response time of the whole system with the better reliability and availability. The fault level of service that are being done by the algorithm which is increases the throughput of the whole system of cloud environment and keeping all the customer get know all that idea of happening in the system which increased the availability. As per our proposed work get into consideration there is a bit of improvement we saw with the existing work. Further, the future work could be done with the some real life scenario.

References

1. Jhawar, Ravi, Vincenzo Piuri, and Marco Santambrogio. "Fault tolerance management in cloud computing: A system-level perspective." *Systems Journal, IEEE* 7.2 (2013): 288-297.
2. Poola, Deepak, Kotagiri Ramamohanarao, and Rajkumar Buyya. "Fault-tolerant workflow scheduling using spot instances on clouds." *Procedia Computer Science* 29 (2014): 523-533.

3. Yao, Lin, et al. "Guaranteeing fault-tolerant requirement load balancing scheme based on VM migration." *The Computer Journal* (2013): bxt012.
4. Bala, Anju, and Inderveer Chana. "Fault tolerance-challenges, techniques and implementation in cloud computing." *IJCSI International Journal of Computer Science Issues* 9.1 (2012): 1694-0814.
5. Rimal, Bhaskar Prasad, Eunmi Choi, and Ian Lumb. "A taxonomy and survey of cloud computing systems." *2009 Fifth International Joint Conference on INC, IMS and IDC*. Ieee, 2009.
6. Malik, Sheheryar, and Fabrice Huet. "Adaptive fault tolerance in real time cloud computing." *Services (SERVICES), 2011 IEEE World Congress on*. IEEE, 2011.
7. Jhawar, Ravi, Vincenzo Piuri, and Marco Santambrogio. "A comprehensive conceptual system-level approach to fault tolerance in cloud computing."*Systems Conference (SysCon), 2012 IEEE International*. IEEE, 2012.
8. Vishwanath, Kashi Venkatesh, and Nachiappan Nagappan. "Characterizing cloud computing hardware reliability." *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010.
9. Lu, Kuan, et al. "Fault-tolerant Service Level Agreement lifecycle management in clouds using actor system." *Future Generation Computer Systems* 54 (2016): 247-259.
10. Randles, Martin, David Lamb, and A. Taleb-Bendiab. "A comparative study into distributed load balancing algorithms for cloud computing." *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. IEEE, 2010.
11. Clark, Christopher, et al. "Live migration of virtual machines." *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005.
12. Deng, Jing, Meikang Qiu, and Gang Wu. "Fault tolerant data collection in heterogeneous intelligent monitoring networks." *Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on*. IEEE, 2010.
13. Wen, Hui, et al. "Effective load balancing for cloud-based multimedia system." *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*. Vol. 1. IEEE, 2011.
14. Zhang, Zehua, and Xuejie Zhang. "A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation."*Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*. Vol. 2. IEEE, 2010.

15. Randles, Martin, David Lamb, and A. Taleb-Bendiab. "A comparative study into distributed load balancing algorithms for cloud computing." *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. IEEE, 2010.

16. Zhu, Xiaomin, Xiao Qin, and Meikang Qiu. "QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters." *Computers, IEEE Transactions on* 60.6 (2011): 800-812.

17. Qiu, Meikang, et al. "A novel energy-aware fault tolerance mechanism for wireless sensor networks." *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*. IEEE, 2011.

18. Wickremasinghe, Bhathiya, Rodrigo N. Calheiros, and Rajkumar Buyya. "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications." *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010.

19. Koslovski, Guilherme, et al. "Reliability support in virtual infrastructures."*Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010.

20. Ayari, Narjess, et al. "Fault tolerance for highly available internet services: concepts, approaches, and issues." *Communications Surveys & Tutorials, IEEE* 10.2 (2008): 34-46.

21. Ardagna, Claudio A., et al. "A model-based approach to reliability certification of services." *Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on*. IEEE, 2012.

22. Costa, Pyramo, et al. "On the performance of Byzantine fault-tolerant MapReduce." *Dependable and Secure Computing, IEEE Transactions on*10.5 (2013): 301-313.

23. Das, Pritam, and Pabitra Mohan Khilar. "VFT: A virtualization and fault tolerance approach for cloud computing." *Information & Communication Technologies (ICT), 2013 IEEE Conference on*. IEEE, 2013.

24. Kutlu, Mucahid, Gagan Agrawal, and Orhan Kurt. "Fault tolerant parallel data-intensive algorithms." *High Performance Computing (HiPC), 2012 19th International Conference on*. IEEE, 2012.

25. Wu, Liying, Bo Liu, and Weiwei Lin. "A dynamic data fault-tolerance mechanism for cloud storage." *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on*. IEEE, 2013.

26. Fu, Yongkang, and Bin Sun. "A scheme of data confidentiality and fault-tolerance in cloud storage." *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*. Vol. 1. IEEE, 2012.

27. Egwutuoha, Ifeanyi P., and Shiping Cheny. "Energy Efficient Fault Tolerance for High Performance Computing (HPC) in the Cloud." *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013.

28. Ganesh, Aman, M. Sandhya, and Subramaniam Shankar. "A study on fault tolerance methods in Cloud Computing." *Advance Computing Conference (IACC), 2014 IEEE International*. IEEE, 2014.

29. Naeem, Tayyiba, and Sarmad Sadik. "Optimizing performance and fault tolerance through cloud based adaptive replication for mobile applications."*Systems and Informatics (ICSAI), 2014 2nd International Conference on*. IEEE, 2014.

30. KUSHWAH, VIRENDRA SINGH, SANDIP KUMAR GOYAL, and PRIUSHA NARWARIYA. "A SURVEY ON VARIOUS FAULT TOLERANT APPROACHES FOR CLOUD ENVIRONMENT DURING LOAD BALANCING."

31. Jhawar, Ravi, and Vincenzo Piuri. "Fault tolerance and resilience in cloud computing environments." *Computer and Information Security Handbook,*(2013): 125-141.

Corresponding Author:

Senthilnathan Palaniappan[1] *,

Email: *senthil.me.psg.it@gmail.com*