Improving Cloud Performance through Performance Based Load Balancing Approach

Er. Surbhi Sharma[1], Er. Intiyaz Ahmad[2], Er. Sourav Mirdha[3]

[1, 2]*M.Tech. Student, Computer Science & Engineering, International Institute of Engineering & Technology, Samani, Kurukshetra, Haryana, India*

[3]*Assistant Professor, Computer Science & Engineering, International Institute of Engineering & Technology, Samani, Kurukshetra, Haryana, India*

Abstract - *Among the major issues of cloud computing, load balancing is the critical issue. It can be achieved through task scheduling, resource management, task resource mapping, efficient virtualization and also by avoiding fault and handling the situation of fault. Fault tolerance is also one of the critical issues. Major work associated with fault tolerance is its detection in advance followed by recovery from it. To tackle this issue, different researchers have given different methodologies. Quality of service provided by CSP can be improved by providing desired resources well in time with minimization of response, service time and failure.*

In this paper, authors have tried to improve the cloud performance through load balancing with fault tolerance. Fault handler, redundancy and check pointing have been used to implement fault tolerance (reactive and proactive). This removes the faulty node and does not make them available for task assignment till its recovery. Also while distributing load among nodes, success ratio and past load data is also considered. This has improved the quality of service as task is getting mapped with that node whose success rate is more and present load is less.

***Key Words*: Cloud Computing, Load Balancing, Fault Tolerance, Virtualization, Cloudsim**

1. INTRODUCTION

Cloud computing has recently emerged as a new form of the utility-based computing paradigm for hosting and delivering hardware and software "as services". It provides its users with the illusion of infinite computing and storage resources which are potentially available on-demand from anywhere and anytime. Cloud computing is attractive since it eliminates the requirement for its users to plan ahead for provisioning, by allowing IT enterprises to start from the small and to increase resources only when there is a rise in service demand. However, despite of this, the development of techniques to make cloud computing effective is currently at its infancy, with many issues still to be addressed [1].

"A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers" [2].

Essential characteristics of cloud computing are as follows:

- **On-demand self-service:** a consumer can autonomously provision computing capabilities (e.g., computing power, storage space, network bandwidth), that is without requiring human interaction with the respective provider(s);
- **Rapid elasticity:** the above capabilities may be dynamically resized in order to quickly scale up (to potentially unlimited size) or down in according to the specific needs of the consumer [3].

1.1 Architecture of Cloud System

A cloud system, that is a system which adopts the cloud computing paradigm, can be characterized by its architecture and the services it offers. The architecture of a cloud computing system is usually structured as a set of layers. A typical architecture of a cloud system is shown in Figure 1 (from [4]). At the lowest level of the hierarchy there is the hardware layer, which is responsible for managing the physical resources of the cloud system, such as servers, storage, network devices, power and cooling systems. On the top of the hardware layer, resides the infrastructure layer, which provides a pool of computing and storage resources by partitioning the physical resources of the hardware layer by means of virtualization technologies. Built on top of the infrastructure layer, the platform layer consists of operating systems and application frameworks. The purpose of this layer is to minimize the burden of deploying applications directly onto infrastructure resources by providing support for implementing storage, database and business logic of cloud applications. Finally, at the highest level of the hierarchy there is the application layer, which consists of cloud applications.

For what regards services implemented on top of a cloud computing system, they can be provided in three modality, according to the abstraction level of the capability provided and the service model of providers [2]:

- **Infrastructure as a Service (IaaS)**, which comprises services to allow its consumers to request computational, storage and communication resources on-demand, thus enabling the so called "pay-per-use" paradigm whereby consumers can pay for exactly the amount of resource they use (like for electricity or water). The consumers can use the provided resources to deploy and run arbitrary software; however, the management and control of the underlying cloud infrastructure is possible only by the provider. An example is Amazon EC2 [5].

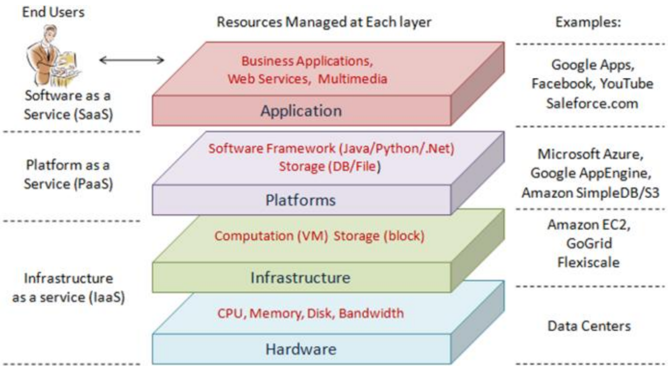

Fig -1: The architecture of a cloud system (from [4])

- **Platform as a Service (PaaS)**, which comprises high-level services providing an independent platform to manage software infrastructures, where consumers (i.e., developers) can build and deploy particular classes of applications using programming languages, libraries, and tools supported by the provider. Usually, consumers don't manage or control the underlying infrastructure (such as servers, network, storage, or operating systems), which can only be accessed by means of the high-level services provided by the provider. An example is Google App Engine [6].
- **Software as a Service (SaaS)**, which comprises specific end-user applications running on a cloud infrastructure. Such applications are delivered to consumer as a network service (accessible from various client devices, ranging from desktop computers to smart phones), thus eliminating the need to install and run the application on the consumer's own computers and simplifying maintenance and support. Consumers don't manage or control the underlying infrastructure and application platform; only limited user-specific application configurations are possible. An example is Salesforce.com [7].

The traditional approach to deploy a cloud system is a public computing system. However, other deployment models are possible which differentiate each other's by variations in physical location and distribution. For instance, the following models are taken from NIST [2]:

- **Public cloud**: the cloud infrastructure is provisioned for open use by the general public and is made available in a "pay-per-use" manner;
- **Private cloud**: the cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple users;
- **Community cloud**: the cloud infrastructure is provisioned for exclusive use by a specific community of users from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations);
- **Hybrid cloud:** the cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by technology that enables data and application portability. A typical example is when a private cloud is temporarily supplemented with computing capacity from public clouds, in order to manage peaks in load (also known as "cloud-bursting") [3].

2. LOAD BALANCING & FAULT TOLERANCE

Load balancing can be defined as the process of task distribution among multiple computers, processes, disk, or other resources in order to get optimal resource utilization and to reduce the computation time. Load balancing is an important means to achieve effective resource sharing and utilization. In general, load balancing algorithms can be divided into following three types[8]:

- **Centralized approach:** In this approach, a single node is responsible for managing the distribution within the whole system.
- **Distributed approach:** In this approach, each node independently builds its own load vector by collecting the load information of other nodes. Decisions are made locally using local load vectors. This approach is more suitable for widely distributed systems such as cloud computing.
- **Mixed approach**: A combination between the two approaches to take advantage of each approach [9].

Fault tolerance is an approach where a system continues to work properly even if there is a fault. There are number of fault tolerant techniques are available but still fault tolerance in cloud computing is a difficult task. Because of the wide spread infrastructure of cloud and the increasing demand of services, an efficient fault tolerant technique for cloud computing is essential. But due to its virtualization and internet based service providing behavior, fault tolerance in cloud computing is still a major problem. The main fault tolerance issues in cloud computing are detection and recovery. Fault tolerance mechanism can be implemented at task and work flow level. Fault tolerance mechanism can be divided into two categories [10]:

- Proactive Fault Tolerance
- Reactive Fault Tolerance

Proactive Fault Tolerance:

We try to identify the components which may cause fault and replace them in advance. Some of the commonly used techniques based upon this theory are as follows:

- **Preemptive Migration:** It depends upon the feedback mechanism where system is consistently analyzed.
- **Self Healing:** This is automatically used to handle the failure situation when many instances of the same application are running.
- **Software Rejuvenation:** In this methodology system reboots itself after certain period of time with clean state [11].

Reactive Fault Tolerance:

This type of policies comes in action after occurrence of failure and tries to minimize the effect of failure. Techniques based upon this policy are as follows:

- **Rescue workflow:** In this technique, system will keep on working until it becomes impossible to move forward.
- **Task Resubmission:** This is the most commonly used technique where failed task is resubmitted from the beginning.
- **Task Migration:** After failure, pending task may be migrated to other machines.
- **Check Point:** When a task fails, it is allowed to restart from the last entry done for check point purpose [12, 13].

3. PROPOSED WORK

The proposed load balancing model has used the logic of reactive fault tolerance. Success ratio is assigned to all virtual modes. In the beginning it is .5, while its maximum value is 1. A virtual node becomes eligible for selection if success ratio is lying in (0, 1]. If it is not lying in this interval then that node is not eligible for selection. Diagrammatical representation of proposed approach is shown in figure 1.

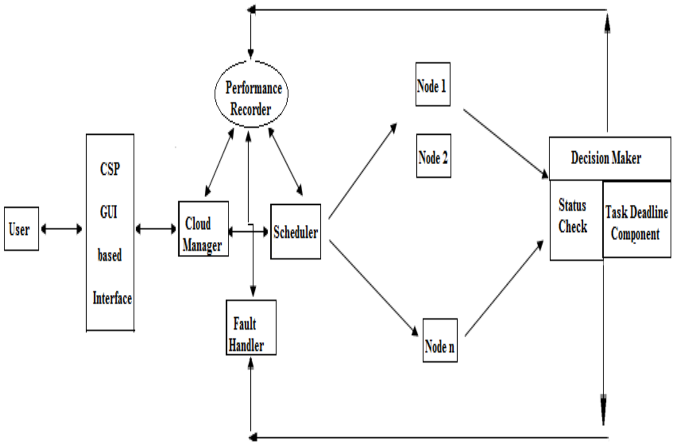

Fig -1: Proposed Approach

Steps of the proposed approach are as follows:

1. User interacts with the CSP through the provided graphical interface.
2. CSP forwards the user request to cloud manager (CM). It maintains the Performance record (PR) table which stores the following entries:
   a. Id of virtual node
   b. Id of associated physical machines
   c. Success ratio of virtual node
   d. Task assignment counter
   e. No of times node has given the successful results (a)
   f. No of times tasks has been assigned to node (b)
3. CM forwards the request towards scheduler which does the load balancing activity. Scheduler access the PR table to assign the task to that VM whose SR is good and present load is less.
4. Whenever a node get fails, fault handler comes in to action. It updates the record of nodes performance in PR table and either restart the server or calls scheduler to transfer the pending task.
5. Execution results are transferred to decision maker module (DM). Through status checker (SC), it gets the information about the status of all virtual machines. DM checks the deadlines of the tasks through Task Deadline Component (TDC).
   a. If both SC & TDC for a VM results in success then its SR is incremented and PR table is updated. SR=a++/b++;
   b. If SC results in fail, then fault handler is called to handle the situation. SR=a/b++;
   c. If SC does not return in fail but TDC results in fail, then its SR is decremented and PR table is updated
   d. DM maintains the list of all those VM who's SC & TDC results in success. Highest SR value of VM is considered as checkpoint for further executions.

Proposed work in the algorithmic form is as follows:
Algorithm LBFT ( )
{
Identify the different available virtual machines;
$V = \{V_1, V_2, \ldots\ldots.. V_n\}$
// Set of available virtual machines
For (i=1 to n)
{
$SR(V_i)=.5$
// Initially Success ratio of all virtual machine is same
Store the following info in the performance record table for each VM;
i. Id of virtual node
ii. Id of associated physical machines
iii. Success ratio of virtual node
iv. Task assignment counter

v. No of times node has given the successful results, its initial value is 1 (a)
vi. No of times tasks has been assigned to node, its initial value is 2 (b)

}
While (Task is there in the data center)
{

- Calculate Priority=success ratio/load for each virtual machine in the performance record table. If load is zero then Priority=success ratio;
- Sort the performance record table on the basis of Priority;
- Select the highest Priority virtual machine from the priority table;
- Assign the task to the selected VM;
- Update the performance record table.

If (status checker of the machine is not fail and task is completed before deadline)
{
Value of SR is updated,
SR=a++/b++;
}
Else if (status checker returns in fail task is completed before deadline)
{
Value of SR is updated,
SR=a/b++;
Fault handler is called to handle fault situation;
}
Else
{
Value of SR is decremented;
}
}
Decision maker maintains the list of all those VM whose status checker & task deadline controller results in success. Highest SR value of VM is considered as checkpoint for further executions.
}

Algorithm Fault_handler(id of virtual machine)
{

- Recalculate the success ratio of received virtual machine;
- Transfer the pending task to other VM using the same approach;

}

4. SIMULATOR AND RESULTS

We can analyze the performance of any load balancing algorithm by actually testing it in cloud environment on various parameters. But it is very costly and difficult to manage the cloud environment only for experiment purpose. So there is a need of simulator to test the load balancing algorithm in cloud environment.

We have used Cloudsim simulator which is free and open source software available at http://www.cloudbus.org/CloudSim/. It is a code library based on Java. This library can be directly used by integrating with the JDK to compile and execute the code. For rapid applications development and testing, Cloudsim is integrated with Java-based IDEs (Integrated Development Environment) including Eclipse or NetBeans. Using Eclipse or NetBeans IDE, the Cloudsim library can be accessed and the cloud algorithm can be implemented [14].

To analyze the variation of success ratio for different virtual machines, we have considered three different virtual machines having initial success ratio .5. We have analyzed the performance of three virtual machines 10 times. Same set of tasks are assigned to all virtual machines. In chart 1, 2 and 3 analysis of success ratio for three virtual machines is given. It has been found that success ratio of virtual machine 1 is almost continuously increasing while for virtual node 2 it is following a random walk pattern. For virtual machine 3 success ratio is decreasing for first half while in second half it is increasing.

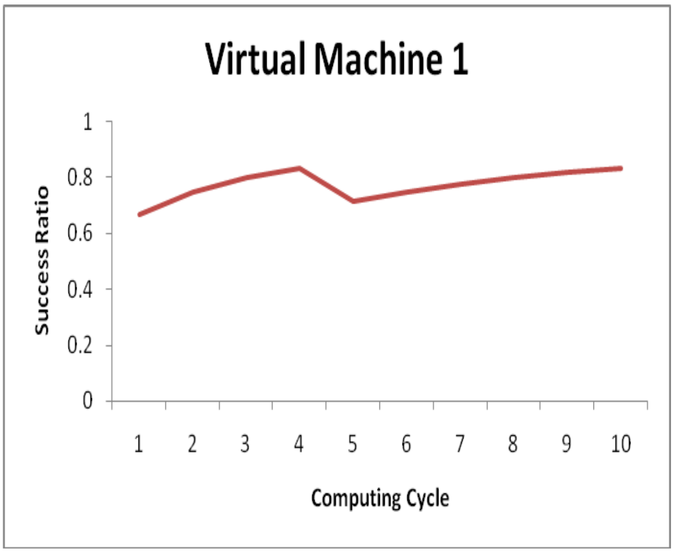

Chart -1: Success ratio analysis of virtual node 1

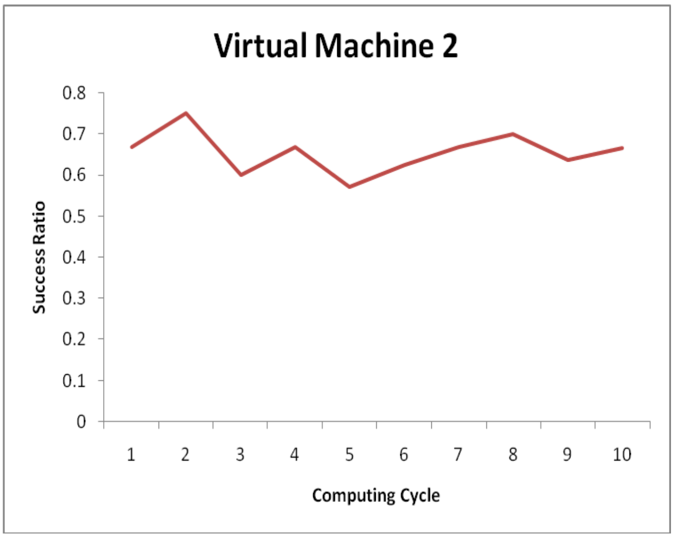

Chart -2: Success ratio analysis of virtual node 2

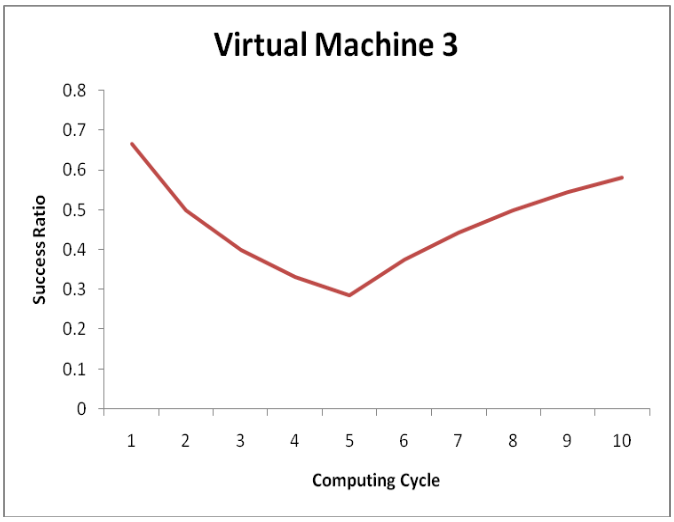

Chart -3: Success ratio analysis of virtual node 3

Table -1: Node Selection Procedure

Sr. No.	Task Deadline	Virtual Machine 1				Virtual Machine 2				Virtual Machine 3				Selected Node
		Status	Deadline	Finish Time	Success Ratio	Status	Deadline	Finish Time	Success Ratio	Status	Deadline	Finish Time	Success Ratio	
Start	-	-	-	-	0.5	-	-	-	0.5	-	-	-	0.5	-
1	1700	Success	Success	1600	0.667	Success	Success	1602	0.667	Success	Success	1610	0.667	1
2	1602	Success	Success	1600	0.75	Success	Success	1602	0.75	Success	Fail	1610	0.5	2
3	1601	Success	Success	1600..8	0.8	Success	Fail	1602	0.6	Success	Fail	1611	0.4	1
4	1605	Success	Success	1601	0.833	Success	Success	1603	0.667	Success	Fail	1611	0.333	1
5	1600	Success	Fail	1602	0.714	Success	Fail	1603	0.571	Success	Fail	1612	0.286	-
6	1900	Success	Success	1602	0.75	Success	Success	1604	0.625	Success	Success	1612	0.375	1
7	1700	Success	Success	1602	0.778	Success	Success	1604	0.667	Success	Success	1612	0.444	1
8	2100	Success	Success	1604	0.8	Success	Success	1604	0.7	Success	Success	1613	0.5	1
9	1700	Success	Success	1603	0.818	Fail	Fail	-	0.636	Success	Success	1613	0.545	1
10	2000	Success	Success	1604	0.833	Success	Success	1605	0.666	Success	Success	1614	0.583	1

Table 1 shows the execution of proposed algorithm for three virtual machines and same set of tasks for 10 times. Arbitrarily task deadlines are assigned. Initially, all the three virtual machines have same success ratio .5. So during first execution, any virtual machine can be selected randomly. In the second execution, two machines have the same success ratio, so among them any one can be selected randomly. In this way, algorithm ensures that every time task will be mapped with the best available virtual machine. In case status of machine become fail, them immediately fault handler is called. Pending tasks are transferred to other virtual machines using the same strategy. Maximum success ratio is updated after execution of every cycle by the highest success ratio of the virtual machine.

Chart 4 shows the comparison of three virtual machines on the basis of number of average task completion time. It has been found that VM1 shows the best performance and VM3 shows the worst performance while performance of VM2 lies between VM1 & VM3. Same kind of relationship has been found when we have compared the performance of three VM on the basis of number of tasks completed before deadline out of 10. VM1 has completed maximum number of tasks while VM3 has completed the minimum number of tasks while VM2 performance lies between VM1 & VM3. This result has been shown graphically in chart 5.

So it can be concluded that priority calculated on the basis of success ratio and load is good scale to select the appropriate VM.

5. CONCLUSIONS

In this paper, author has presented a new load balancing approach by imparting the concept of fault tolerance, success ratio and present load. Success ratio for every virtual machine is calculated based upon its past performance. Based upon the success ratio and present load, priority of each virtual machine is calculated which becomes the deciding factor for selection of virtual machine. Also, on failure of any virtual machine, pending tasks are transferred to other machines. As we are considering the past performance of virtual machine, while mapping task with it, so this makes our approach fault tolerant. As less performing virtual machines have low priority and less chance of selection. So in this way we have added proactive and reactive fault tolerance feature with the proposed load balancing approach.

To improve the proposed approach, we can embed the mechanism of load transfer from overloaded virtual machine to under loaded virtual machine. Also we can embed the resource allotment logic with the proposed load balancing approach.

REFERENCES

[1] A. Jain and R. Kumar, "A Taxonomy of Cloud Computing," International Journal of Scientific and Research Publications. vol. 4(7), Jul. 2014, pp. 1-5.

[2] P. Mell and T. Grance, "The NIST definition of Cloud computing: Recommendations of the national institute of standards and technology," Special Publication 800-145, NIST, Sep 2011.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,". Future Generation Computer Systems, vol. 25(6), 2009, pp. 599–616.

[4] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," Journal of Internet Services and Applications, vol. 1(1), 2010, pp. 7–18.

[5] Amazon elastic compute cloud (EC2). Available: http://aws.amazon.com/ec2.

[6] Google AppEngine: Run your web apps on Google's infrastructure.Available:http://code.google.com/appengine/

[7] Salesforce.com. Available: http://www.salesforce.com.

[8] A. Jain and R. Kumar, "A Multi Stage Load Balancing Technique for Cloud Environment." International Conference on Information Communication and Embedded Systems (ICICES), Feb 2016, pp. 1-7.

[9] A. Khiyaita, M. Zbakh, H. El Bakkali and D. El Kettani, "Load balancing cloud computing: State of art," in Proceedings of 2012 National Days of Network Security and Systems (JNS2), 20-21 April 2012, pp. 106-109.

[10] A. Bala and I. Chana, "Fault Tolerance- Challenges, Techniques and Implementation in Cloud Computing," IJCSI International Journal of Computer Science Issues, vol. 9(1),, January 2012.

[11] R. Jhawar, V. Piuri, and M. D. Santambrogio, "Fault Tolerance Management in Cloud Computing: A System Level Perspective," IEEE InternationalSystems Journal, vol. 7(2), June 2013, pp. 288-297.

[12] I. P. Egwutuoha., S. Chen, D. Levy, and B. Selic, "A Fault Tolerance Framework for High Performance Computing in Cloud," IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, May 2012, pp. 709-710.

[13] D. Poola, K. Ramamohanarao, and R. Buyya, "Fault Tolerant Workflow Scheduling Using Spot Instances on Clouds," 14th ELSEVIER International Conference on Computational Science (ICCS), 2014, pp. 523-533.

[14] RN. Calheiros, R. Ranjan, A. Beloglazov, CA. De Rose, R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and experience. vol. 41(1), Jan 2011, pp.23-50.