*International Journal of Engineering Science Invention (IJESI)*
*ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726*
*www.ijesi.org ||Volume 8 Issue 04 Series. I || April 2019 || PP 43-47*

# Significance of Rebalancing Engine in Multi Cloud Computing System

B.J. D Kalyani[1], Dr. Kolasani Ramchand H Rao[2]
[1](Research Scholar, Department of Computer Science and Engineering, Acharya Nagarjuna University,Guntur, India)
[2](ramkolasani@gmail.com)
Corresponding Author: B.J. D Kalyani

***Abstract :*** *Multi cloud environment enables the user to provide special features, more features that are not available in a single cloud and user has liberty of options in selecting cloud service provider not to be bonded to a single cloud. Hence enterprises are looking for multi cloud strategy to deploy their workloads efficiently. But user preferences are not meet at the early stage of or at request stage of multi cloud but they are satisfied at a later stage i.e. at the execution stage due to elastic nature of the infrastructure clouds. This can be achieved through rebalancing policies to replace instances in lower-preferred clouds with instances in higher-preferred clouds are needed to meet user preferences with rebalance engine. This paper proposes need for rebalancing policies, role of rebalancing engine and methodology for auto scaling.*

***Keywords -*** *Multi Cloud Computing System (MCCS), Rebalancing Engine, Auto Scaling*

---

Date of Submission: 29-03-2019 Date of acceptance: 09-04-2019

---

## I. INTRODUCTION

MCCS [1] is recommended to reduce latency and to increase response time of an enterprise used by the multiple geographic markets. MCCS supports better disaster recovery system by establishing cross cloud server and storage resources in separate locations. Multi cloud user requests can be expressed in two forms: 1) in terms of absolute numbers of instances needed in selected clouds, e.g., R= {32 instance in cloud A, 8 instances in cloud B}; 2) in terms of total numbers of instances and preferred ratios, e.g., R= {40 instances total; 80% in cloud A, 20% in cloud B}. However, such requests may lead to situations where a deployment cannot be satisfied, at least initially. For example, instead of matching the request R= {32 instances in cloud A, 8 instances in cloud B}, we may have 24 instances in cloud A (which may not be able to launch additional instances) and, thus, end up with 16 instances in cloud B. Therefore, as the environment adapts, additional instances should be launched in cloud A whenever possible and instances in cloud B should be terminated until the users' preferences are met. Thus rebalancing engine plays an important role in the architecture of MCCS as in figure 1.

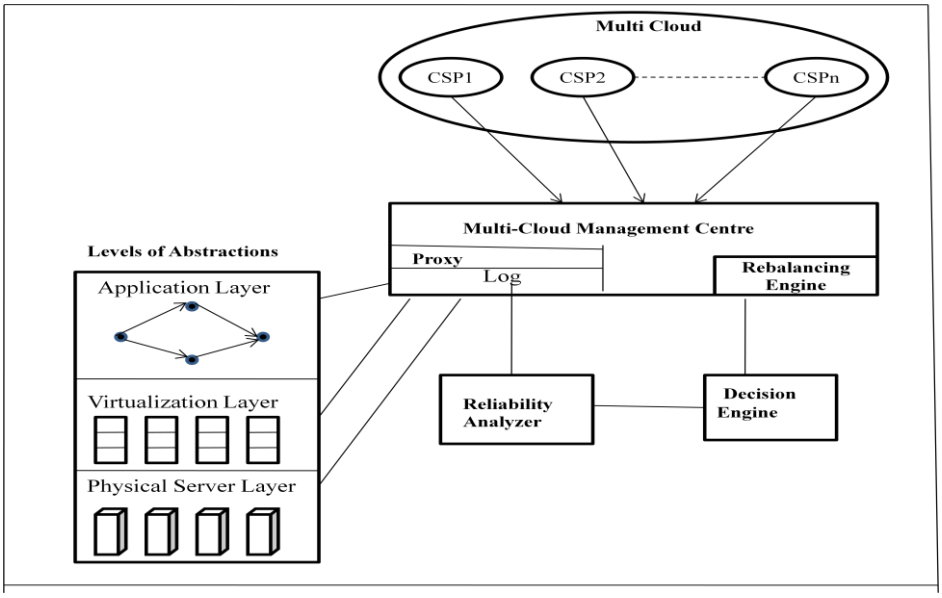


**Figure 1:** MCCS Architecture

## II. Relationship Between Levels Of Abstractions And Re

The top layer of levels of abstraction is application layer used to deploy the applications in MCCS with the help of below layers infrastructure. By considering the reliability as standard metric the relationship between levels of abstraction and RE are explored. The multi cloud VM reliability tester tool [2] supports reliability engineering methods that apply to VM's of multi cloud infrastructure. The fundamental approach which is used in multi cloud reliability tester is depends on a stress test that assess response time of the VMs. The idea of this stress test [3] is to experimentally estimate failure rates of virtual machines using OpenStack instance. These failure rates are analyzed by statistical methods of data analysis in order to fit a statistical model that is suitable for distribution of failure rates for measurement. This evaluated model is then used to predict failure rates that could encounter during normal OpenStack operation. In order to verify the reliability model, one must test the predicted values in a second stress test and validate if they are still a good fit in the test set of VMs. The framework of the multi cloud reliability tester tool is designed in a three step process.

1. The Computation Independent Model (CIM) is designed to implement the above mentioned four step process. Then the software elements which implement every process step are combined to the model.
2. As the software elements which deploys each process step are identified, then the abstract view of logical software elements that facilitates implementation of on demand functionality are combined to the framework and is referred as a Platform Independent Model (PIM).
3. The PIM is now shifted to a model of system components (software components, hardware nodes) that is tangible enough to allow for building the software framework. This software framework is called a Platform Specific Model (PSM), as it specifies the technologies that are needed to implement the logical functional units of the software.

VM reliability Tester tool can conduct the above mentioned steps with the python code and is associated with genchart.py to demonstrate the performance of VMs of this layer as in figure 2.

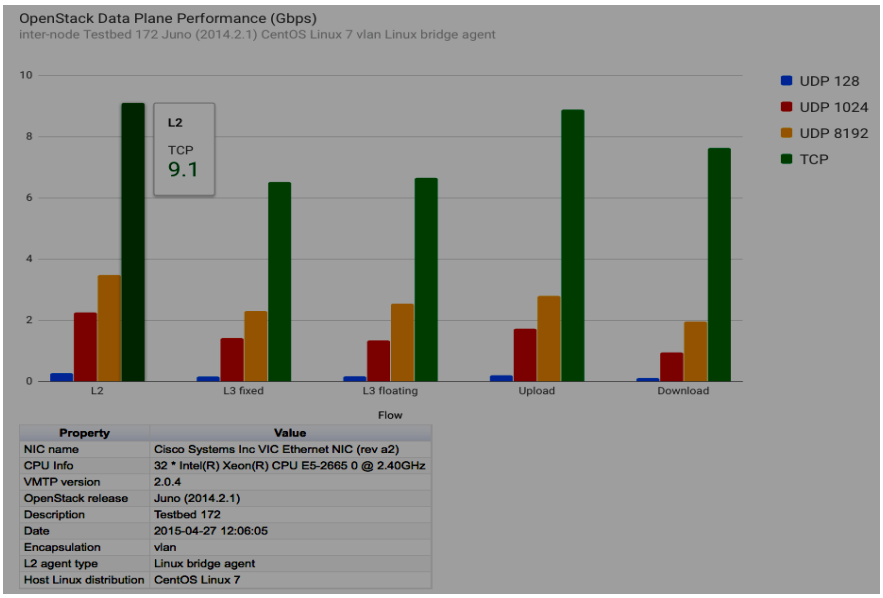


**Figure 2**: Performance of VMs in Virtualization layer

Physical server layer help organizations create administrative and resource boundaries [4] between applications. For MCCS to be reliable, there is a need to identify the failures like hardware, service, software or resource, their causes and manages them to improve their reliability. The failure analysis can be done with the help of three models namely self-governing fault model, an associated fault model and an infected fault model. As the term resilience denotes as the ability to supply and maintain an acceptable level of service in spite of a variety of faults and attacks, the resiliency [5] is assessed using following algorithm.

```
Algorithm: Resiliency Assessment
Receive cloudlet and examine its priority
Sort the waiting queue in decreasing priority order
        if priority is HIGH then
                Calculate number of servers
                Schedule task
                Represent the state of server
        else
                Suspend the task
                Set server to 1 and Repeat from step 2
        end if
Select 0<= R Needed <= 1
        for i= 1 to number of servers
                While (R Actual < R Needed) do
                        if P requested > P available then
                                Increase P available
                        end if
                        if M requested > M available then
                                Increase M available
                        end if
                        if H requested > H available then
                                Increase H available
                        end if
                        if B requested > B available then
                                Increase B available
                        end if
                end while
                switch on faultmodel
                        case self: get F app and r
                                break;
                        case Associate: get F app and r
                                break;
                        case Infected: get F app and r
                                break;
                end case
                S app = S app + F app
                S r = S r + r
end for
```

The physical server layer reliability despite of faults can be modeled in figure 3.

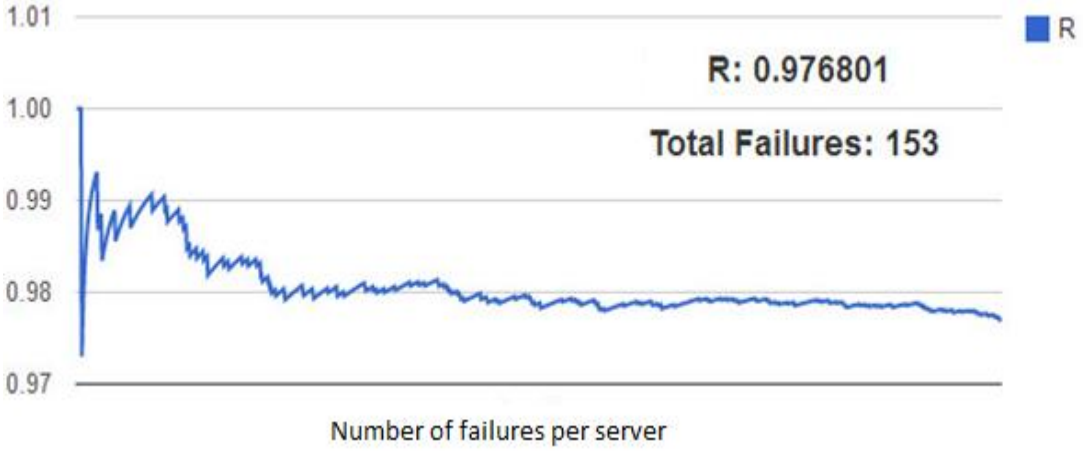


**Figure 3**: Reliability of Physical Server Layer

## III. Framework Of RE

The framework of rebalance engine establishes master-worker logical relationship with decision engine of proposed architecture. It employs workers across various multi cloud infrastructures and dynamically balances this deployment based on user-defined preferences. The framework of rebalance engine is represented in Figure 8.1 and consists of four main components: (1) a workload management system (2) sensors to monitor

demand, (3) policies to scale the number of deployed instances up or down, and (4) an auto-scaling service to enforce the selected policy.

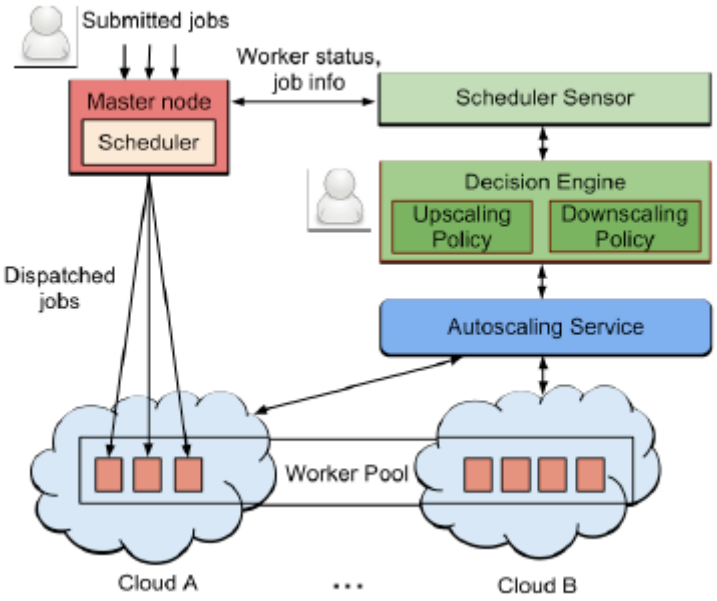


**Figure 4:** Framework of RE

## IV. Decision Engine

Decision engine is associated with auto scaling which performs up scaling [7] and down scaling policies. Up scaling attempts to launch instances on clouds with a higher preference, however, if such clouds are unavailable then up scaling will deploy instances on clouds with lower preferences. Down scaling is implemented with three policies like:

1. Expedient Idle (EI): This rebalancing policy waits until excess instances in less desired clouds are idle (i.e., not running jobs according to information from the sensor) and then terminates them.
2. Potency-Offline (PO): This policy is similar to EI but excess instances are terminated gracefully, that is, jobs are allowed to complete before the instances are terminated.
3. Hostile Policy (HP): This policy sacrifices work cycles and discards partially completed jobs in order to satisfy requests in the shortest amount of time possible. This policy terminates excess instances [8] even if those instances are currently running jobs. To minimize overhead associated with job re-execution, this policy proceeds in termination from instances with jobs that have been running for the least amount of time to instances with jobs that have been running for a longer time.

By considering workload execution time, convergence time, workload overhead and excess cost as basic metrics the proposed rebalancing policies can be compared as in figure 5.

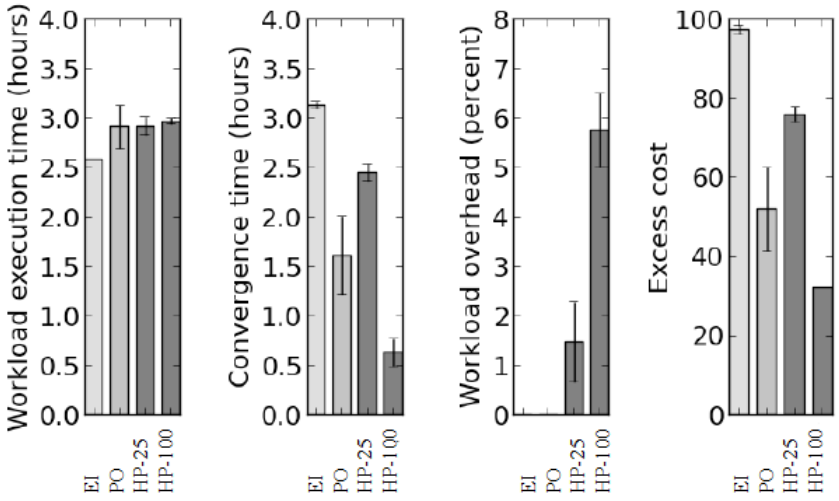


**Figure 5:** Comparison of proposed policies

## V. Conclusion

This paper focused on the vital component of MCCS architecture name rebalancing engine, describes the importance of it. The frame work of RE and gives an overview of its logical relationship with levels of abstractions of MCCS architecture. This paper proposed three rebalancing polices and are compared using various performance metrics.

## References


**[1].** d. Alpert and d.l. bitzer, advances in computer-based education. Science, vol. 167, pp. 1,582-1,590, mar. 1970.

[2]. Celesti, A.; Tusa, F.; Villari, M.; Puliafito, A , Improving Virtual Machine Migration in Federated Cloud Environments, Evolving Internet (INTERNET), 2010 Second International Conference on , vol., no., pp.61-67, 20-25 Sept. 2010.

[3]. K. Pandey and N. K. Goyal, **Early Software Reliability Prediction**, Studies in Fuzziness and Soft Computing 303, DOI: 10.1007/978-81-322-1176-1_2, Springer India 2013.

[4]. David Bernstein, Erik Ludvigson, Krishna Sankar, Steve Diamond, and Monique Morrow. **Blueprint for the intercloud - protocols and formats for cloud computing interoperability.** In Mark Perry, Hideyasu Sasaki, Matthias Ehmann, Guadalupe Ortiz Bellot, and Oana Dini, editors, ICIW, pages 328–336. IEEE Computer Society, 2009.

[5]. Mariam Rahmani , Azad Azadmanesh, **Exploitation of Quantitative Approaches to Software Reliability,** Department of Computer Science, University of Nebraska at Omaha, Technical Report No. cst-2011-002 , December 2011.

[6]. Amir Vahid Dastjerdi and Rajkumar Buyya, **An Autonomous Reliability-aware Negotiation Strategy for Cloud Computing Environments**, 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 978-0-7695-4691-9/12 $26.00 © 2012 IEEE DOI 10.1109/CCGrid.2012.101.

[7]. Zhao Wu, Naixue Xiong, Yannong Huang, Qiong Gu, Chunyang Hu, Zhongbo Wu, and Bo Hang, **A Fast Optimization Method for Reliability and Performance of Cloud Services Composition Application**, Journal of Applied Mathematics Volume 2013, Article ID 407267, DOI http://dx.doi.org/10.1155/2013/407267.

[8]. Navendu Jain & Nachiappan Nagappan, **Understanding Network Failures in Data Centers: Measurement, Analysis and Implications**, IBM Research Hawthorne, NY Dec. 1, 2011.


B.J. D Kalyani" Significance of Rebalancing Engine in Multi Cloud Computing System" International Journal of Engineering Science Invention (IJESI), Vol. 08, No. 04, 2019, PP 43-47