Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 45 (2015) 832 – 841

Procedia Computer Science

International Conference on Advanced Computing Technologies and Applications (ICACTA-2015)

Autonomous Agent Based Load Balancing Algorithm in Cloud Computing

Aarti Singh[a], Dimple Juneja[b], Manisha Malhotra[a]*

[a]*Maharishi Markandeshwar University, Mullana, 133205, India*
[b]*Dronacharya Institute of Management and Technology, Kurukshetra, 136119, India*

Abstract

Cloud Computing revolves around internet based acquisition and release of resources from a data center. Being internet based dynamic computing; cloud computing also may suffer from overloading of requests. Load balancing is an important aspect which concerns with distribution of resources in such a manner that no overloading occurs at any machine and resources are optimally utilized. However this aspect of cloud computing has not been paid much attention yet. Although load balancing is being considered as an important aspect for other allied internet based computing environments such as distributed computing, parallel computing etc. Many algorithms had been proposed for finding the solution of load balancing problem in these fields. But very few algorithms are proposed for cloud computing environment. Since cloud computing is significantly different from these other types of environments, separate load balancing algorithm need to be proposed to cater its requirements. This work proposes an Autonomous Agent Based Load Balancing Algorithm (A2LB) which provides dynamic load balancing for cloud environment. The proposed mechanism has been implemented and found to provide satisfactory results.

*Keywords:*Agents; Cloud Computing; Dynamic Load Balancing.

1. Introduction

* Corresponding author. Tel.: +91-8427465352
E-mail address: mmanishamalhotra@gmail.com

doi:10.1016/j.procs.2015.03.168

The cloud computing is a distributed internet based paradigm, designed for remote sharing and usage of different resources and services like storage, computational capabilities and applications etc. with high reliability over the large networks. However, due to dynamic incoming requests, dynamic resource allocation is required in it. This inherent dynamism in cloud computing requires efficient load balancing mechanisms. Load balancing concerns distribution of resources among the users or requests in uniform manner so that no node is overloaded or sitting idle. Like in, all other internet based distributed computing tasks, load balancing is an important aspect in cloud computing. In the absence of load balancing provision, efficiency of some overloaded nodes can sharply degrade at times, leading to violation of SLA. In traditional distributed computing, parallel computing and grid computing environments load balancing algorithms are categorized as static, dynamic or mixed scheduling algorithms based on their nature [6] where:

a)Static Load Balancing Algorithm is suitable for small distributed environments with high Internet speed and ignorable communication delays.

b)Dynamic Load Balancing Algorithm focuses on reducing communication delays and execution time and thus are suitable for large distributed environments.

c)Mixed Load Balancing Algorithm focuses on symmetrical distribution of assigned computing task and reducing communication cost of distributed computing nodes.

Based on above categorization, cloud computing clearly falls under the second category. It means balancing load in cloud computing environment requires focusing on dynamic load balancing algorithms. In traditional distributed environments process migration is less expensive due to small process granularity whereas in CC environment, process migration is expensive due to high granularity of data involved. Thus cloud computing environment requires a load balancing algorithm which could cater to dynamic service demands of users while providing optimized load balancing. Following parameters are available in literature for measuring efficiency of a load balancing algorithm in CC environment [4]:

i.*Reliability*: The algorithm must be reliable, since process failure while transferring job from one location to other may lead to increased waiting time and customer dissatisfaction.

ii.*Adaptability:* Algorithm must be capable of adapting the dynamically changing user requests and provide task allocation in minimal amount of time.

iii.*Fault Tolerance:* The algorithm must ensure fault tolerance, so that in case of a problem in the system complete load balancing mechanism does not stop working.

iv.*Throughput:* the algorithm must ensure increased throughput at minimal expense. If a load balancing algorithm doesn't increase system throughput, it defeats its own purpose.

v.*Waiting Time:* Algorithm should minimize wait time of a task for allocation of resources to it.

Next subsection elaborates major components of a dynamic load balancing algorithm.

1.1Components of Dynamic Load Balancing Algorithms

Literature review highlighted that a load balancing algorithm has five major components [6] as discussed below:

Table1: Components of Load Balancing

Sr. No.	Policy	Function
1	**Transfer Policy**	This policy is responsible to determine when a task should be transferred from one node to the other node.
2	**Selection Policy**	It focuses on selecting the processor for load transfer so that the overall response time may be improved.
3	**Location Policy**	It determines the availability of required resources for providing services and makes a selection based on location of resources.
4	**Information Policy**	This policy acquires workload related information about the system such as nature of workload and average load on each node. It is also responsible for exchanging the information from one node to another, along with method of exchange and amount of the information to be exchanged. For exchanging load information of a node one of the following three methods may be adopted: ***i. Broadcast Approach:*** If it is assumed a broadcast communication medium is available, then a load exchange is done whenever the load node changes. ***ii. Global System Load:*** Whenever a node does not acknowledges the revert from another node in a complementary stage, it presumes that all nodes are overloaded.

		iii. Polling Approach: In the idle (or overloaded) state of a node the neighbours or randomly polled nodes send the request to get the information.
5	**Load Estimation Policy**	It determines the total workload of a node in a system. Need for balancing load is triggered by this policy.

The remainder of the paper is organized as follows: Section 2 contains a review of the related work. Section 3 describes the proposed work. Implementations and results are analyzed in section 4. Finally conclusion of work is given in section 5.

Next section provides the review of relevant literature in cloud computing.

2. Review of Existing Load Balancing Algorithms in Cloud Computing

Load balancing is one of important problems of heterogeneous computer networks. To address this problem, many centralized approaches have been proposed in the literature but centralization has proved to raise scalability tribulations. Randles et. al [8] provided a comparative analysis of various dynamic load balancing algorithms (Honeybee foraging, Biased Random Sampling, and Active Clustering). Their analysis has highlighted that honeybee algorithm has maximum throughput with increased system diversity as compared to other two algorithms. The honeybee algorithm is motivated from the behaviour of biological bees that move in search of their food. Similarly in load balancing there are virtual servers offering virtual services. Every server requiring services calculates the profit and posts it on its advert board. The servers interested in serving the request also calculate their profit and compare it with the colony profit. If case of high colony profit interested server serves the current virtual server otherwise returns to the scout behaviour i.e. to choose another server randomly.

Hu. et. al. [5] proposed genetic algorithm based scheduling mechanism for load balancing among virtual machines. This mechanism selects the least loaded virtual machine for load transfer and optimizes the high migration cost. However due to large number of virtual machines and frequent service requests in the data centre, there is chance of inefficient service scheduling. Xu et. al. [4] introduced a model for load balancing in public cloud by using game theory. This algorithm is based on cloud portioning. They divided the cloud into three categories idle, normal and overloaded on the basis of load degree. Zero load degree represents an idle cloud whereas if it lies between zero and highest value then the cloud status is normal otherwise the cloud is overloaded. Here method of selecting range for load degree has been left unaddressed. Wang et. al. [13] has proposed two static algorithms in cloud environment. One is for Opportunistic load balancing in which incoming tasks received by a node have minimum execution time which is calculated by service manager. Second is Load Balance Min Min which improves the resource utilization by maintaining the load balance. However both these algorithm are not suitable for CC as they do not support dynamic environments.

Osman et al. [19] proposed a system to migrate legacy and network application by providing a virtualization layer on top of the operating system and transferring a process group. They achieve lower downtime of service, but still use stop-and-copy approach. Nakai et. al [20], introduced an approach for client-based load distribution that adaptively changes the fraction of the load that each client submits to each service replica to minimize overall response times. Bhaskar et. al. [2] proposed a mechanism working in two phases. In first phase it finds the CPU utilization and memory required for each instance and also finds the memory available for each virtual machine. In second phase, it compare the available resources with required resources, if required resources are available then proceed further otherwise discard the request. Drawback of this mechanism is that it lacks in scalability. Xu et. al [15] has introduced an agent based model using decision theory. The aim of this model is to reduce the computational cost involved in load balancing. The migration concept used in this architecture transfers the load from overloaded nodes to under loaded nodes.

From the literature it has been observed that there are some drawbacks such as static nature of load balancing algorithms, lack of scalability and reliability. Further from analysis of literature it being observed that artificial intelligent mechanism such as genetic algorithm, honeybee algorithm, game theory and intelligent agents had been employed for load balancing in cloud computing, which highlights that researchers have found them suitable for such applications and there is scope of employing them further. Thus there is strong need for an efficient load balancing mechanism in cloud computing.

Next section elaborates the proposed work.

3. Proposed Work

From the literature review it is clear that limited work has been done for load balancing in cloud computing environment and those existing mechanisms do have limitations that need to be addressed. Thus there is need of an algorithm which can offer maximum resource utilization, maximum throughput, minimum response time, dynamic resource scheduling with scalability and reliability. This work proposes an **a**utonomous **a**gent based **l**oad **b**alancing algorithm (A2LB) to address above issues. Whenever a VM becomes overloaded, the service provider has to distribute the resources in such a manner that the available resources will be utilized in a proper manner and load at all the virtual machines will remain balanced. A2LB mechanism comprises of three agents: Load agent, Channel Agent and Migration Agent. Load and channel agents are static agents whereas migration agent is an ant, which is a special category of mobile agents. The reason behind deploying ants is their ability to choose shortest/best path to their destination. Ant agents are motivated from biological ants which seek a path from their colonies to the food source. While doing so they secrete a chemical called pheromone on ground [16] thus leaving a trail for other colleagues to follow. However this chemical evaporates with time. Initially the ants start searching a food source randomly, thus they may follow different paths to the same source, however with passage of time, density of pheromone on the shortest path increase and thus all follower ants start following that path resulting in increase of pheromone density even further. An appealing property of ants is that they move from source to destination for collecting desired information or performing a task but they do not necessarily come back to their source rather they destroy themselves at the destination only thereby reducing unnecessary traffic on the network. Since load balancing in CC would require searching for under loaded servers and resources, ant agents suit the purpose and fulfill it appropriately without putting additional burden on network. Description of various agents deployed in A2LB is as follows:

Load Agent (LA): It controls information policy and maintains all detail of a data centre. The major work of a load agent is to calculate the load on every available virtual machine after allocation of a new job in the data centre. This agent is supported with table termed as VM_Load_Fitness table.

VM_Load_Fitness table: It is used for maintaining record of specifications of all virtual machines of a data centre. It contains virtual machine id, status of its memory consumed along with CPU utilization, fitness value and load status of all VMs. Its structure is shown below in Table 2.

Table 2: VM_Load_Fitness Table

Virtual Machine_ID	Memory(μ)Used	CPU Utilization (λ)	Fitness Value (ν)	Load Status
VM_1	μ_1	λ_1	ν_1	Normal
VM_2	μ_2	λ_2	ν_2	Critical
|	|	|		|
VM_n	μ_n	λ_n	ν_n	Normal

Where μ is the percentage of memory used, λ is the CPU utilization percentage and ν is the fitness value for a virtual machine.

Channel Agent (CA): It controls the transfer policy, selection policy and location policy. On receiving the request from load agent, the channel agent will initiate some migration agents to other data centres for searching the virtual machines having similar configuration. It also keeps the record of all messages received from these agents in response table in sorting order which is as given below:

Table 3: Response Table

Migration Agent_ID	Destination Data Centre_ID	Response Received	Migration Agent Status
MA_1	DC_Id_1	*A	Live
MA_2	DC_Id_2	**NA	Destroy
|	|	|	|
MA_n	DC_Id_n	A	Live

*A: <Applicable> Found Similar Configuration
**NA: <Not Applicable> Not Found Similar Configuration

Migration Agent (MA): These agents are initiated by channel agent. It will move to other data centres and communicate with load agent of that data centre to enquire the status of VMs present there, looking for the desired configuration. On receiving the required information it communicate the same to its parent channel agent. Afterwards, it will stay at destination location, waiting for self-destroy message from parent CA channel agent. The status of migration agent may be live or destroyed based on its applicability.

Figure 1 given below provides high level view of proposed mechanism.

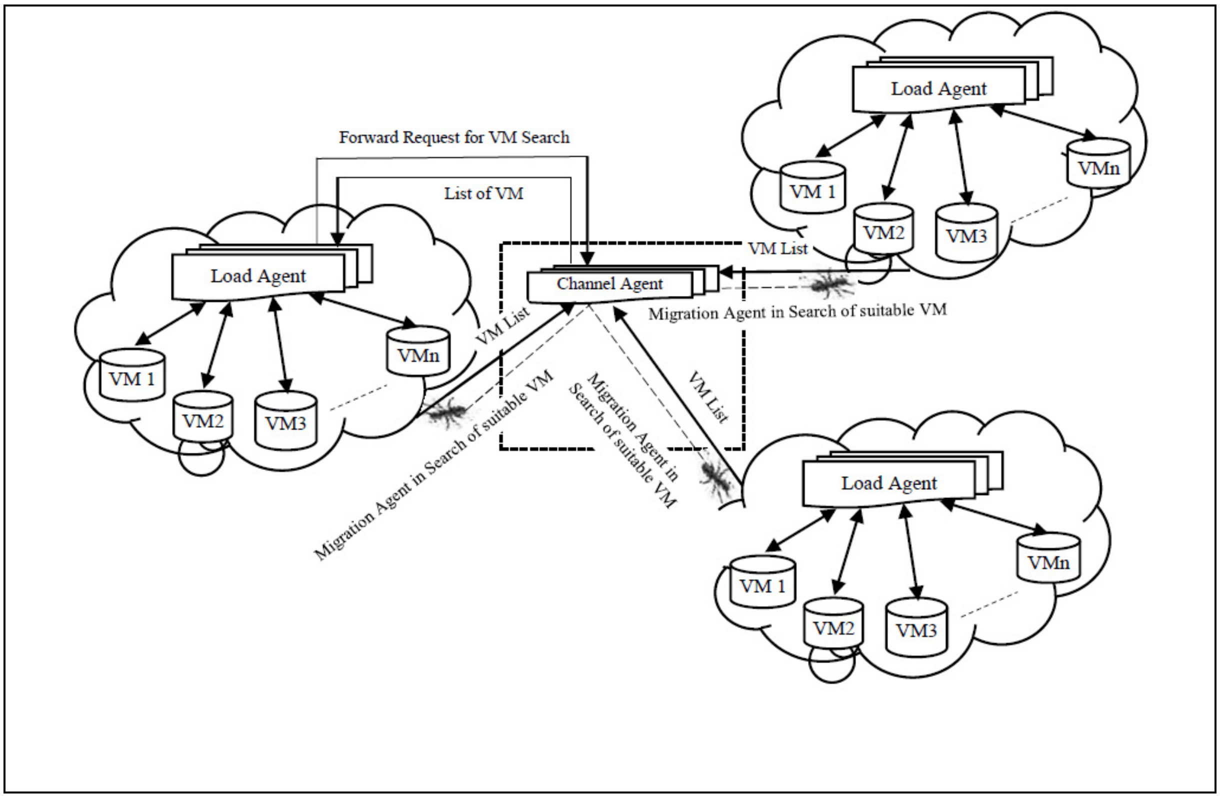

Forward Request for VM Search
List of VM
Load Agent
Channel Agent
VM List
Migration Agent in Search of suitable VM
VM 1
VM2
VM3
VMn
VM List
Migration Agent in Search of suitable VM
Migration Agent in Search of suitable VM
VM List
Load Agent
VM 1
VM2
VM3
VMn
Load Agent
VM 1
VM2
VM3
VMn

Fig.1: High level view of A2LB

Load agent acts proactively for calculating load status of various VMs available in a DC. Periodically it determines the workload of virtual machines in terms of available memory, available CPU utilization, and expected response time. Afterwards it calculates the fitness value of each virtual machine which directly proportional to the memory of a machine and can be evaluated by equation 1, 2 and 3:

$$\mu_{available} = \mu_{total} - \mu_{used} \tag{1}$$

$$\upsilon(\%) = \frac{\mu_{available}}{\mu_{total}} \times 100 \tag{2}$$

The percentage of fitness values will gives the status of a virtual machine.

$$v = \begin{cases} \leq 25\% \, Critical \quad Allocation \\ > 25\% \, Normal \quad Allocation \end{cases} \tag{3}$$

Now whenever a request arrives in a data centre, after allocating resources to it load agent will update VM_Load_Fitness table to reflect present status of all VMs. For this load agent calculates percentage of μ and λ since these factors affect processing of incoming requests. Based on value of μ available, fitness value (ν) for each node is generated. As long as ν of a node is greater than a threshold (25%), in this case VM status is normal. As and when fitness value of a VM becomes less than or equal to threshold value, load balancing needs to be performed. Load agent on observing critical status of a VM will intimate and send the specification of that VM to the channel agent. Then channel agent will initiate the migration agents to other data centres for searching the virtual machines having similar specifications. Migration agents being ants will travel one way. On reaching a destination data centre, migration agent will first send an acknowledgement message to its parent channel agent. Afterwards it will check with load agent of that data centre for availability of virtual machines having similar configuration as desired. If no such VM exists at that data centre, migration agent sends a <Not-Applicable> message back to its parent channel agent and waits for <self_destroy> instruction from it. However, if one or more VMs having desired configuration are found, migration agent further checks their μ and ν sends it to channel agent.

On receiving responses from various migration agents, channel agent maintains them in response analysis table as shown below:

Table 4: Response_Analysis Table

Migration Agent_ID	Destination Data Centre_ID	Virtual Machine_ID	Fitness Value (ν)	Response Time
MA_1	DC_Id_1	VM_1	ν_1	t_1
MA_2	DC_Id_2	VM_2	ν_2	t_2
|	|	|	|	|
MA_n	DC_Id_n	VM_n	ν_n	t_n

Response_Analysis table contains responses of live migration agents only. Each row indicates status of a desired VM configuration, along with its ν. Additionally time of response for each migration agent is also recorded so as to measure freshness of recorded data. It is necessary since once a migration agent sends a response, it is not necessary that same VM would be used for balancing load, but still that information may be used at some time in near future, thus live migration agents are instructed to periodically update status of their concerned VMs to parent channel agents.

On receiving a new request in the data centre, the load agent will map the specification with the available virtual machines. If the fitness value of a VM is normal, load agent proceeds future for allocation otherwise load agent will call channel agent for perspective data centres having VMs with similar configuration for load balancing. At this time, channel agent scans response analysis table and finds <MA_{id}, DC_{id},, VM_{id}>for matching the request. If more than one suitable record is found, it picks the record with largest ν. Channel agent then communicates with corresponding migration agent to confirm current ν of VM under consideration. On receiving response from migration agent, channel agent again analyses all suitable VMs, if still same VM has highest ν, it passes that record to load agent for further load balancing, otherwise channel agent again communicate with migration agent for new suitable VM. Then channel agent would send this information to load agent for further processing.
Algorithms for various agents employed in this mechanism are as follows.

3.1 Algorithms

Algorithms of various agents deployed in proposed framework are given below in figure 2(a), 2(b) and 2(c) respectively.

```
Migration_Agent ( )
Input: $VM_{configurations}$ fromChannel_Agent($VM_{initial}$)
Output: Search similar VM from other Datacenters
{
Accept $VM_{configurations}$ fromchannel_Agent($VM_{initial}$);
Search a Data_center;
Check VM_Load_Table;
If (found)
Return (A);
Else
Return (NA);
On_receiving (self_destroy);
Kill ($MA_i$);
}
```

Fig.2(a): Algorithm of Migration Agent

```
Load_ Agent( ):
Input: Receive request from user;
Output: Allocate_ resources_with_A2LB;
Case I:
{
If (VM_Load_Table==empty( ) )
Then allocate_requested_resources;
Maintain_VM_Load_Table;
$\mu_{available} = \mu_{total} - \mu_{used}$
$\upsilon(\%) = \frac{\mu_{available}}{\mu_{total}} \times 100$
If ($\upsilon$ >25) then
{
allocation_status:=Normal;
}
Else
{
allocation_status:=Critical;
initiateChannel_Agent($VM_{initial}$);
}
Case II:
If (VM_Load_Table≠ empty)
Scan VM_Load_Table;
If(Load_Status($VM_i$) ==Critical)
{
Call Channel_Agent($VM_{Load\_Balance}$);
Receive <$DC_{id}$,$VM_{id}$forload_ transfer>;
Transfer_request to $DC_{id}$;
}
Else
Allocate_request to $VM_{id}$;
Update VM_Load_Table;
}
```

Fig.2(b): Algorithm of Load Agent

```
Channel_Agent($VM_{initial}$):
Input:Receive$VM_i$configuration from Load_Agent;
Output:Response_Analysis_Table, VM_id;
On_Receiving$VM_i$ fromLoad_Agent( );
Initiate_Migration_agent( );
Receive acknowledgement from Migration_Agents( );
Maintain_Response_Table;
If (response == NA)
{
Send self_destroy($MA_i$);
}
Else
{
Receive $\upsilon$ ($MA_i$);
Maintain Response_Analysis_Table;
Periodically update Response_Analysis_Table;
}

Channel_Agent($VM_{Load\_Balance}$):
On incoming request;
{
Scan Response_Analysis_Table;
Prepare list of matching $VM_i$;
L1: for i=1 to n
Large=0;
If ($\upsilon$ ($VM_i$) >large)
Large=$\upsilon$ ($VM_i$);
$\upsilon$ old= $\upsilon$ i;
Update message to $MA_i$;
If ($\upsilon$ old= = $\upsilon$ i )
Return (<$DC_i$, $VM_i$>) to Load_Agent;
Else
Goto L1;
}
```

Fig.2(c): Algorithm of Channel Agent

Next section discusses the implementation of proposed algorithm with results.

4. Implementation and Results

This section implements the proposed A2LB algorithm by using java technology. The implementation is done on a small scale environment by taking number of parameters are shown in table 5:

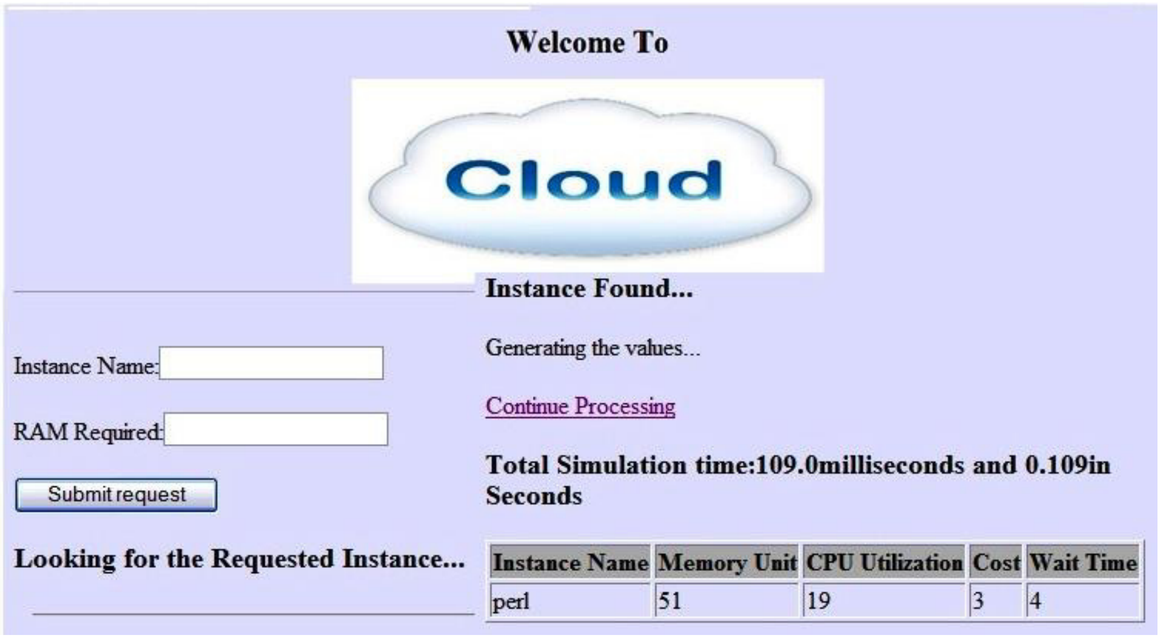
Table 5: Parameter Table

Sr. No.	Parameter Name	Parametric Value
1	No. of Data Centre	3
2	No. of Virtual Machines per data centre	4
3	No. of Instances per Virtual Machine	6
4	Memory Unit	In GB
5	Cost	$
6	Wait Time	In Milliseconds(ms)
7	Number of Runs	15

With all these parameters, implementation is done by two ways: in first case when requested virtual machine is found with normal status and allocation takes place. In second case virtual machine is in critical state, then load balancing takes place. Further second case is implemented by using A2LB algorithm and without A2LB algorithm, also to observe performance of proposed mechanism.

Table 6: Three cases of Implementation

Case I	Normal Allocation (Under loaded VM)
Case II	Critical Allocation (Overloaded VM) (*With A2LB Algorithm*)
Case III	Critical Allocation (Overloaded VM) (*Without A2LB Algorithm*)



Instance Found...

Generating the values...

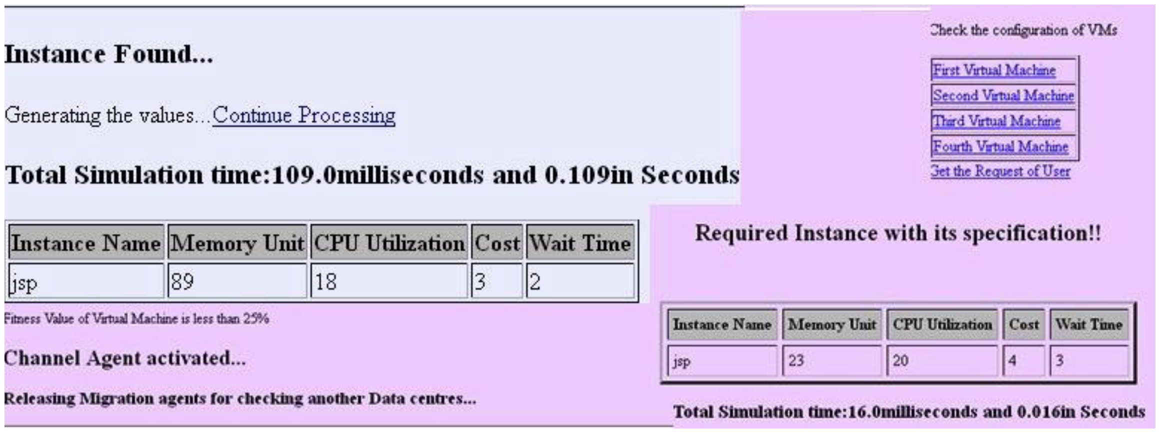
Total Simulation time:109.0milliseconds and 0.109in Seconds

Looking for the Requested Instance...

Instance Name	Memory Unit	CPU Utilization	Cost	Wait Time
perl	51	19	3	4

Fig.3: Cloud Interface and Total Response Time in case I

Figure 3 shows the cloud interface in which the cloud user can demand the required instances. All these executions are done 15 number of times. Initially the implementations is done when instance found in its own data centre i.e all demanded virtual machines are under loaded hence allocation status is normal. It takes total 109ms (response time) for completion of whole execution in first run. In second case when status of requested virtual machine was found critical then A2LB algorithm was applied for load balancing. Figure 4 shows when the fitness value of virtual machine will become less than 25% after allocation then automatically load agent will call the channel agent and executes proactively A2LB. Channel agent will have the prior possible solutions for more incoming requests.


Instance Found...

Generating the values...Continue Processing

Total Simulation time:109.0milliseconds and 0.109in Seconds

Instance Name	Memory Unit	CPU Utilization	Cost	Wait Time
jsp	89	18	3	2

Fitness Value of Virtual Machine is less than 25%

Channel Agent activated...

Releasing Migration agents for checking another Data centres...

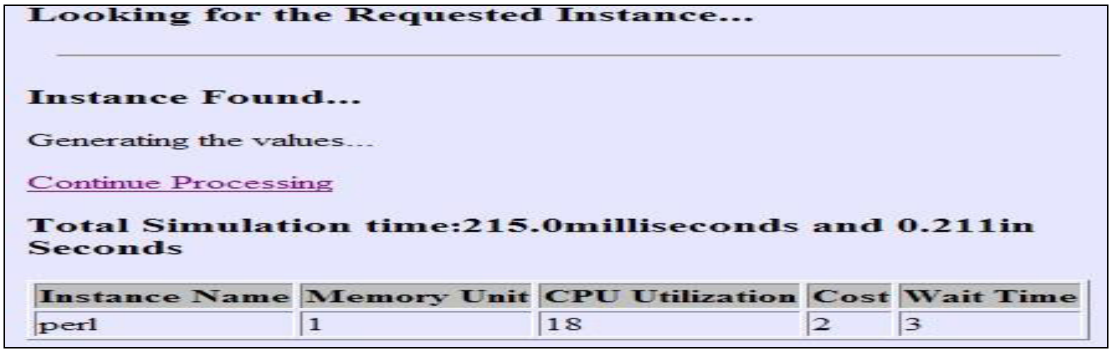
Required Instance with its specification!!

Instance Name	Memory Unit	CPU Utilization	Cost	Wait Time
jsp	23	20	4	3

Total Simulation time:16.0milliseconds and 0.016in Seconds

Fig.4: Total Response Time in case II

When the fitness value becomes less than its threshold, channel agent gets activated and will search the virtual machine having same configuration. It took only 16milliseconds additional time. So total time will become 125ms. Figure 5 describes the last case when demanded virtual machine becomes overloaded and provider will search the similar virtual machine after receiving the request i,e without applying the A2LB algorithm. It takes 215 milliseconds time which is almost double than case I and II.


Looking for the Requested Instance...

Instance Found...

Generating the values...

Continue Processing

Total Simulation time:215.0milliseconds and 0.211in Seconds

Instance Name	Memory Unit	CPU Utilization	Cost	Wait Time
perl	1	18	2	3

Fig.5: Total Response Time in case III

Figure 6 illustrates the comparison between three cases. It is clear from the chart that in all executions, response time remains double in case III. It takes 212ms average response time. Even if we compare the graph of case I and case II there is no more difference in deviations. The average response time is 97ms in case I and 113ms in case II. So it is clear that A2LB takes optimum time when virtual machine becomes overloaded. From this implementation it is revealed that A2LB algorithm provides desired results. Next section concludes this work.

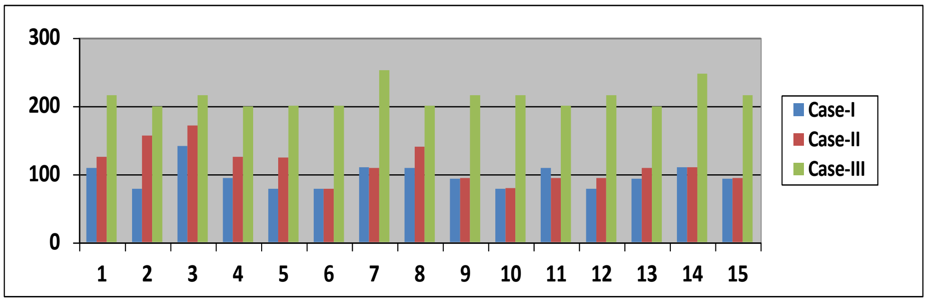


Fig.6 Comparison Between three Cases

5.Conclusion

This work focuses on load balancing in cloud computing environment. Load balancing in cloud computing has been ignored, but rapid growth in number of cloud users has raised demand for load balancing mechanisms. This work has proposed an autonomous agent based load balancing mechanism which provides dynamic load balancing for cloud environment. Major contribution of this mechanism is proactive load calculation of VM in a DC and whenever load of a VM reaches near threshold value, load agent initiates search for a candidate VM from other DCs. Keeping information of candidate VM beforehand, reduces service time. Result obtained through implementation proved that this algorithm works satisfactorily.

References

1.Bhaskar. R, Deepu. S.R and Dr. B.S. Shylaja (2012, September). Dynamic Allocation Method for Efficient Load Balancing in Virtual Machines for Cloud Computing Environment. Advanced Computing: An International Journal (ACIJ), 3(5), pp. 53-61.

2.Dr. PK Sinha, SR Dhore,(2010),Multi-Agent Optimized Load balancing Using Spanning Tree for Mobile Services, International Journal Of Computer Application , 1(6).

3. G. Xu, J. Pang, X. Fu. (2013, Feb). A Load Balancing Model Based on Cloud Partitioning for the Public Cloud. Tsinghua Science and Technology.[Online].*18(1),*pp. 34-39.

4.J. Hu, J. Gu, G. Sun, T. Zhao. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment in *Proc. PAAP,* 2010, pp. 89-96.

5.K. B. Mahieddine. An Evaluation of Load Balancing Algorithms for Distributing System A thesis of Doctor of Philosophy submitted in The University of Leeds, School of Computer Studies, October 1991.

6.M. Amar, K. Anurag, K. Rakesh, K. Rupesh, Y. Prashant (2011). SLA Driven Load Balancing For Web Applications in Cloud Computing Environment, Information and Knowledge Management, 1(1), pp. 5-13.

7.M.Randles,D.Lamb,AT.Bendiab. A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing. In *Proc. ICAINAW*, 2010, pp.551-556.

8.R. Ezumalai,G. Aghila, R. Rajalakshmi,(2010, Feb). Design and Architecture for Efficient Load balancing with Security Using Mobile Agents. International Journal of Engineering & Technology(IACSIT). [Online]. 2(1), pp. 149-160.

9.S Jing and K She (2011 April). A Novel Model for Load Balancing in Cloud Data Centre. Journal of Convergence Information Technology. 6(4),pp. 29-38.

10.S. Ray and A.D. Sarkar (2012, October). Execution Analysis of Load Balancing Algorithms in Cloud Computing Environment. International Journal on Cloud Computing: Services and Architecture (IJCCSA). 2(5), pp. 1-13.

11.S. S. Moharana, R. D. Ramesh & D.Powar,(2013, May). Analysis of Load Balancers In Cloud Computing. International Journal of Computer Science & Engineering (IJCSE). [Online]. 2(2), pp.: 101-108.

12.S.C. Wang,K.Q. Yan, W.P.Liao, S.S. Wang. Towards a Load Balancing in a three-Level Cloud Computing Network. In *Proc. ICCSIT,* 2010, pp.108-113.

13.T. Desai, J. Prajapati,(2013, Nov). A Survey of Various Load Balancing Techniques And Challenges In Cloud Computing. International Journal of Scientific & Technology Research, [Online]. 2(11), pp.158-161.

14.Y.Xu, L. Wu, L. Guo, Z.Chen, L.Yang, Z.Shi. An Intelligent Load Balancing Algorithms Towards Efficient Cloud Computing. In *Proc. AAAI Workshop,* 2011, pp. 27-32.

15.Z Zhang and X Zhang. A Load Balancing Mechanism Based on Ant Colony and Complex Network Theory in Open Cloud Computing Federation.I*n Proc. ICIMA,* 2010, pp. 240-243.

16.Z.Chaczko, V. Mahadevan, S.Aslanzadeh, C. Mcdermid. Availability and Load Balancing In Cloud Computing. In *Proc. ICCSM* , 2011, pp.134-140.

17.Clark, C., Fraser, K., Hand, S., Jacob, G.H.Live migration of virtual machines. In: 2nd ACM/USENIX Symposium on Network Systems, Design and Implementation (NSDI), pp. 273–286 (2005).

18.Osman, S., Subhraveti, D., Su, G., Nieh, J. The design and implementation of ZAP: a system for migrating computing environments. ACM SIGOPS Oper. Syst. Rev. **36**(SI), 361–376 (2002).

19.Nakai, A., Madeira, E., Buzato, L.E. Improving the QoS of web services via client-based load distribution. In: Proceedings of the 29th Brazilian Symposium on Computer Networksand Distributed Systems (SBRC2011) (2011).

20.Cardellini, V., Colajanni, M., Yu, P.S. Request redirection algorithms for distributed web systems. IEEE Trans. Parallel Distrib. Syst. **14**(4), 355–368 (2003).

21. A Keren and A Barak (2013 January). Opportunity Cost Algorithms for Reduction of I/O and Interposes Communication Overhead. In a Computing Cluster. *IEEE Trans.* 14 (1), pp. 399-446.