ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

Harmonizing Model in Cloud Computing Environment

B.Sreekanth[1], G.Lokesh[2]

[1]Student,Dept Of Computer Science & Engineering, Vemu Institute Of Technology,Tirupathi , Puthalapattu,Chittoor(Dist),A.P,India

[2]Assistant Professor,Dept Of Computer Science & Engineering, Vemu Institute Of Technology, Puthalapattu , Chittoor(Dist),A.P,India

ABSTRACT: We present a system that uses virtualization technology to assign data center resources dynamically based on request demands and support green computing by optimizing the number of servers in use. We introduce the concept of "skewness" to measure the unevenness in the multidimensional store utilization of a server. By minimizing skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources. We develop a set of heuristics that prevent overload in the system effectively while saving energy used. Trace driven simulation and experiment results demonstrate that our algorithm achieves good performance. This article introduces a better load balance model for the public cloud based on the cloud partition concept with a switch mechanism to choose different strategies for different situations. The algorithm applies the game theory to the load balancing strategy to improve the efficiency in the public cloud environment.

I. INTRODUCTION

We present the design and execution of an automated resource management system that achieves a good balance between the two goals. We make the following contributions: We develop a resource allocation system that can avoid excess in the system effectively while minimizing the number of servers used. We introduce the concept of "skewness" to compute the uneven utilization of a server. By minimizing skewness, we can improve the overall utilization of servers in the face of multidimensional resource constraints. We design a load prediction algorithm that can capture the future resource usages of applications accurately without looking inside the VMs. The algorithm can capture the rising trend of resource usage patterns and help reduce the placement churn significantly.Since the job arrival pattern is not predictable and the capacities of each node in the cloud differ, for load balancing

problem, workload control is crucial to improve system presentation and maintain stability. Load balancing schemes depending on whether the system dynamics are important can be either static and dynamic . Static schemes do not use the system information and are less complex while dynamic schemes will bring additional costs for the system but can change as the system status changes. A dynamic scheme is used here for its flexibility. The model has a main controller and balancers to gather and analyze the information. Thus, the dynamic control has little influence on the other working nodes. The system status then provides a basis for choosing the right load opposite strategy. The load balancing model given in this article is aimed at the public cloud which has numerous nodes with dispersed computing resources in many different geographic locations. Thus, this model divides the public cloud into several cloud partitions. When the environment is very large and complex, these divisions simplify the load balancing. The cloud has a main manager that chooses the suitable partition for arriving jobs while the balancer for each cloud partition chooses the best load balancing strategy.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

II.RELATED WORK

There have been many studies of load balancing for the cloud environment. Load balancing in cloud computing was describe in a white paper written by Adler who introduced the tools and techniques usually used for load balancing in the cloud. However, load balancing in the cloud is still a new problem that needs new architectures to adapt to many changes. Chaczko et al. described the role that load balancing plays in improving the performance and maintaining stability. There are many load balancing algorithms, such as Round Robin, Equally Spread Current Execution Algorithm, and Ant Colony algorithm. Nishant et al.used the ant colony optimization method in nodes load balancing. Randles et al.gave a compared analysis of some algorithms in cloud computing by checking the performance time and cost. They concluded that the ESCE algorithm and throttled algorithm are better than the Round Robin algorithm. Some of the classical load balancing methods are similar to the allocation method

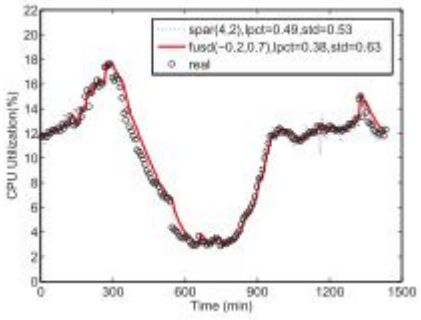
in the in service system, for example, the Round Robin algorithm and the First Come First Served (FCFS) rules. The Round Robin algorithm is used here because it is fairly simple.

III.THE SKEWNESS ALGORITHM

We introduce the concept of skewness to quantify the unevenness in the utilization of multiple resources on a server. Let n be the number of resources we consider and be the utilization of the i th resource. We define the resource skewness of a server p as

$$skewness(p) = \sqrt{\sum_{i=1}^{n} \left(\frac{r_i}{\bar{r}} - 1\right)^2},$$

where $\bar{r}$ is the average use of all resources for server p. In practice, not all types of resources are performance dangerous and hence we only need to consider bottleneck resources in the above calculation. By minimizing the skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources. In the following, we describe the details of our algorithm. Analysis of the algorithm is obtainable in the additional file, which can be



Comparison of SPAR and FUSD.

found on the ComputerSociety Digital Libraryhttp://doi.ieeecomputersociety.org/10.1109/TPDS.2012.283.

3.1 Hot and Cold Spots

Our algorithm executes sometimes to evaluate the resource allocation status based on the predicted future resource demands of VMs. We define a server as a hot spot if the operation of any of its resources is above a hot threshold. This indicates that

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

the server is overloaded and hence some VMs running on it should be migrated away. We define the temperature of a hot spot p as the square sum of its resource utilization beyond the hot threshold:

$$temperature(p) = \sum_{r \in R} (r - r_t)^2,$$

where R is the set of overloaded resources in server p and r is the hot threshold for resource r. (Note that only overloaded resources are considered in the calculation.) The temperature of a hot spot reflects its degree of overload. If a server is not a hot spot, its temperature is zero.We define a server as a cold spot if the utilizations of all its resources are below a cold threshold. This indicates that the server is mostly idle and a probable candidate to turn off to save energy. However, we do so only when the average resource utilization of all actively used servers (i.e., APMs) in the system is below a green computing threshold. A server is actively used if it has at least one VM running. Otherwise ,it is inactive. Finally, we define the warm threshold to be a level of resource utilization that is adequately high to justify having the server running but not so high as to risk becoming a hot spot in the face of temporary fluctuation of application resource demands. Different types of resources can have different thresholds. For example, we can define the hot thresholds for CPU and memory resources to be 90 and 80 percent, respectively. Thus a server is a hot spot if either its CPU usage is above 90 percent or its memory usage is above 80 percent.

3.2 Hot Spot Mitigation

We sort the list of hot spots in the system in descending temperature (i.e., we handle the hottest one first). Our goal is to remove all hot spots if possible. Otherwise, keep their temperature as low as possible. For each server p, we first decide which of its VMs should be migrated away. We sort its list of VMs based on the resulting temperature of the server if that VM is migrated away. We aim to travel away the VM that can reduce the server's temperature the most. In case of ties, we select the VM whose removal can reduce the skewness of the server the most. For each VM in the list, we see if we can find a destination server to accommodate it. The server must not become a hot spot after accepting this VM. Among all such servers, we select one whose skewness can be reduced the most by accepting this VM. Note that this reduction can be negative which means we select the server whose skewness increase the least. If a destination server is found, we record the migration of the VM to that server and update the predicted load of related servers. Otherwise, we move onto the next VM in the list and try to find a destination server for it. As long as we can find a destination server for any of its VMs, we consider this run of the algorithm a success and then move onto the next hot spot. Note that each run of the algorithm migrates away at most one VM from the overloaded server. This does not essentially eliminate the hot spot, but at least reduces its temperature. If it remains a hot spot in the next decision run, the algorithm will repeat this process. It is possible to design the algorithm so that it can migrate away multiple VMs during each run. But this can add more load on the related servers during a period when they are already overloaded. We decide to use this more conservative approach and leave the system some time to react before initiating additional migrations.

3.3 Green Computing

When the resource utilization of active servers is too low, some of them can be turned off to save energy. This is handled in our green computing algorithm. The test here is to reduce the number of active servers during low load without sacrificing performance either now or in the future. We need to avoid oscillation in the system. Our green computing algorithm is invoked when the average utilizations of all resources on active servers are below the green computing threshold. We sort the list of cold spots in the system based on the ascending order of their memory size. Since we need to migrate away all its VMs before we can shut down an underutilized server, we define the memory size of a cold spot as the aggregate memory size of all VMs running on it. Recall that our model assumes all VMs connect to a shared back-end storage. Hence, the cost of a VM live migration is resolute mostly by its memory footprint. This Section in the supplementary file explains why the memory is a good measure in depth. We try to eliminate the cold spot with the lowest cost first. For a cold spot p, we check if we can migrate all its VMs somewhere else. For each VM on p, we try to find a

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

target server to accommodate it. The resource utilizations of the server after accepting the VM must be below the warm threshold. While we can save energy by consolidating underutilized servers, overdoing it may create hot spots in the future. The warm threshold is designed to prevent that. If multiple servers satisfy the above criterion, we prefer one that is not a present cold spot. This is because increasing load on a cold spot reduces the likelihood that it can be eliminated. However, we will accept a cold spot as the destination server if necessary. All things being equal, we select adestination server whose skewness can be reduced the most by accepting this VM. If we can find destination servers for all VMs on a cold spot, we record the sequence of migrations and update the predict load of related servers. Otherwise, we do not migrate any of its VMs. The list of cold spots is also efficient because some of them may no longer be cold due to the proposed VM migrations in the above process. The above consolidation adds extra load onto the related servers. This is not as serious a problem as in the hot spot improvement case because green computing is initiated only when the load in the system is low. Nevertheless, we want to bound the extra load due to server consolidation. We restrict the number of cold spots that can be eliminated in each run of the algorithm to be no more than a certain percentage of active servers in the system. This is called the consolidation limit. Note that we eliminate cold spots in the system only when the average load of all active servers (APMs) is below the green computing threshold. Otherwise, we leave those cold spots there as potential destination equipment for future offloading. This is consistent with our philosophy that green computing should be conducted conservatively.

3.4 Consolidated Movements

The movements generated in each step above are not executed until all steps have finished. The list of actions are then consolidated so that each VM is moved at most once to its final destination. For example, hot spot mitigation may dictate a VM to move from PM A to PM B, while green computing dictates it to move from PM B to PM C. In the actual execution, the VM is moved from A to C directly

IV. SYSTEM MODEL

There are several cloud computing categories with this work focused on a public cloud. A public cloud is based on the standard cloud compute model, with service provided by a service provider. A large public cloud will include many nodes and the nodes in different geographical locations. Cloud partition is used to manage this large cloud. A cloud partition is a subarea of the public cloud with divisions based on the geographic locations. The architecture is shown in Fig.1.The load opposite strategy is based on the cloud partitioning concept. After creating the cloud partitions, the load balancing then starts: when a job arrives at

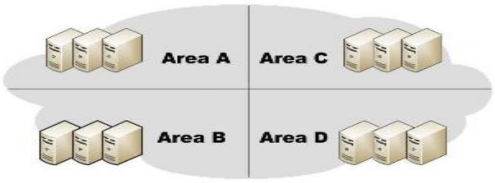


Fig. 1 Typical cloud partitioning.

the system, with the main controller deciding which cloud partition should receive the job. The partition load balancer then decides how to assign the jobs to the nodes. When the load status of a cloud partition is normal, this partition can be talented locally. If the cloud partition load status is not normal, this job should be transferred to another partition. The whole process is shown in Fig.2.

4.1 Main controller and balancers

The load balance solution is done by the main controller and the balancers. The main controller first assigns jobs to the suitable cloud partition and then communicates with the balancers in each partition to refresh this status information.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

Since the main manager deals with information for each partition, smaller data sets will lead to the higher processing rates. The balancers in each partition gather the status in order from every node and then choose the right strategy to distribute the jobs. The relationship between the balancers and the main controller is shown in Fig.3.

4.2 Assigning jobs to the cloud partition

The cloud partition status can be divided into three types:

(1) Idle: When the percentage of idle nodes exceeds α, change to idle status.

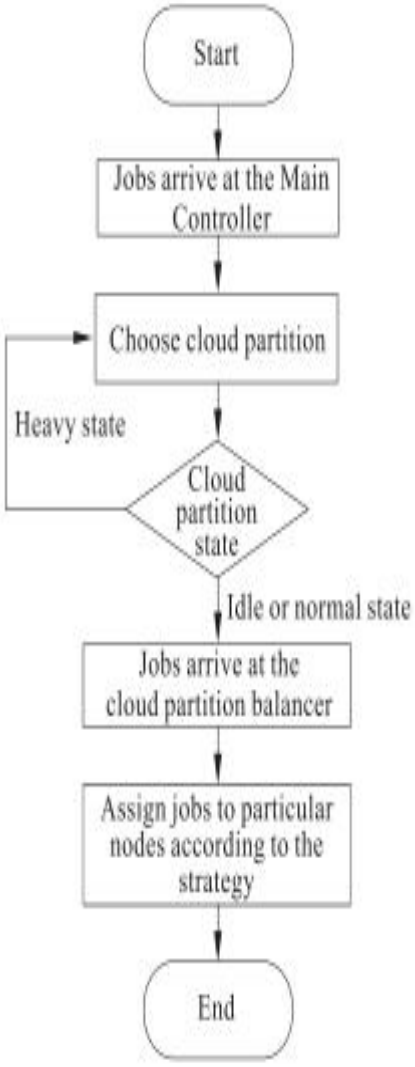


Fig. 2 Job assignment strategy.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

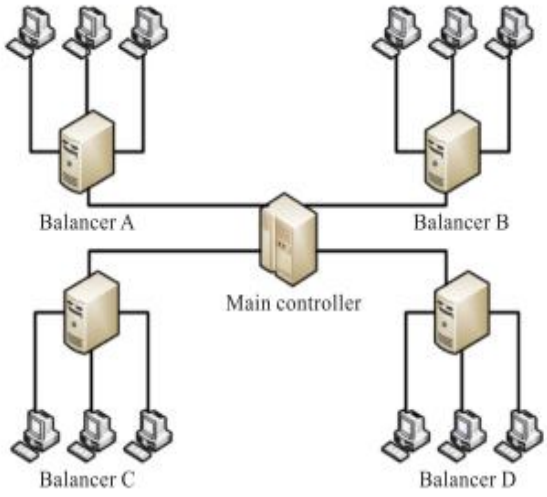


Fig. 3 Relationships between the main controllers, the balancers, and the nodes.

(2) Normal: When the percentage of the normal nodes exceeds β, change to normal load status.

(3) Overload: When the percentage of the overloaded nodes exceeds γ, change to overloaded status. The parameters α, β, and γ are set by the cloud partition balancers. The main controller has to converse with the balancers frequently to refresh the status information. The main controller then dispatches the jobs using the following strategy: When job i arrives at the system, the main controller queries the cloud divider where job is located. If this location's status is idle or normal, the job is handled locally. If not, another cloud partition is found that is not overloaded. The algorithm is shown in Algorithm 1.

4.3 Assigning jobs to the nodes in the cloud partition

The cloud partition balancer gathers load in order from every node to evaluate the cloud partition status. This evaluation of each node's load status is very important. The first task is to define the load degree of

Algorithm 1 Best Partition Searching

```
begin
  while job do
    searchBestPartition (job);
    if partitionState == idle || partitionState == normal then
      Send Job to Partition;
    else
      search for another Partition;
    end if
  end while
end
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

each nodes. The node load degree is related to various static parameter and dynamic parameters. The static parameters include the number of CPU's, the CPU dispensation speeds, the memory size, etc. Dynamic parameters are the memory utilization ratio, the CPU utilization ratio, the network bandwidth, etc. The load degree is computed from these parameters as below:

Step 1 Define a load paramet $F = \{F_1, F_2, \cdots, F_m\}$ with each $F_i (1 \leqslant i \leqslant m, F_i \in [0, 1])$ parameter being either static or dynamic. m
represents the total number of the parameters.

Step 2 Compute the load degree as:

$$\text{Load\_degree}(N) = \sum_{i=1}^{m} \alpha_i F_i,$$

$\alpha_i (\sum_{i=1}^{n} \alpha_i = 1)$ are weights that may differ for different kinds of jobs. N represents the current node.

Step 3 Define estimate benchmarks. Calculate the average cloud partition degree from the node load degree statistics as:

$$\text{Load\_degree}_{\text{avg}} = \frac{\sum_{i=1}^{n} \text{Load\_degree}(N_i)}{n}.$$

The bench mark $\text{Load\_degree}_{\text{high}}$ is then set for different situations based on the $\text{Load\_degree}_{\text{avg}}$.

Step 4 Three nodes load status levels are then defined as:

_ Idle When

$$\text{Load\_degree}(N) = 0,$$

there is no job being processed by this node so the status is charged to Idle.

_ Normal For

$$0 < \text{Load\_degree}(N) \leqslant \text{Load\_degree}_{\text{high}},$$

the node is normal and it can process other jobs.

_ Overloaded When

$$\text{Load\_degree}_{\text{high}} \leqslant \text{Load\_degree}(N),$$

the node is not available and can not receive jobs until high it returns to the normal. The load degree results are input into the Load Status Tables created by the cloud partition balancers. Each balancer has a Load Status Table and refreshes it each fixed period T . The table is then used by the balancers to calculate the partition status. Each partition status has a different load balancing solution. When a job arrives at a cloud partition, the balancer assigns the job to the nodes based on its current load strategy. This strategy is changed by the balancers as the cloud partition status changes.

V.CLOUD PARTITION LOAD BALANCING STRATEGY

5.1 Motivation

Good load balance will improve the presentation of the entire cloud. However, there is no common method that can adapt to all possible different situations. Various methods have been developed in improving existing solutions to resolve new problems. Each exacting method has advantage in a particular area but not in all situations. Therefore, the current model integrates several methods and switches between the load balance method based on the system status. A relatively simple method can be used for the divider idle state with a more complex method for the normal state. The load balancers then switch methods as the status changes. Here, the idle status uses an improved Round Robin algorithm while the normal status uses a game theory based load balancing strategy.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

5.2 Load balance strategy for the idle status

When the cloud partition is idle, many computing resources are available and relatively few jobs are arriving. In this situation, this cloud partition has the ability to process jobs as quickly as possible so a simple load opposite method can be used. There are many simple load balance algorithm methods such as the Random the Weight Round Robin, and the Dynamic Round Robin. The Round Robin algorithm is used here for its simplicity. The Round Robin algorithm is one of the simplest load balancing algorithms, which passes each new request to the next server in the queue. The algorithm does not record the status of each connection so it has no status information. In the regular Round Robin algorithm, every node has an equal opportunity to be chosen. However, in a public cloud, the configuration and the performance of each node will be not the same; thus, this method may overload some nodes. Thus, an improved Round Robin algorithm is used , which called "Round Robin based on the load degree evaluation".

The algorithm is still fairly simple. Before the Round Robin step, the nodes in the load balancing table are ordered based on the load degree from the lowest to the highest. The system builds a circular queue and walks through the queue again and again. Jobs will then be assign to nodes with low load degrees. The node order will be changed when the balancer refreshes the Load Status Table. However, there may be read and write inconsistency at the refresh period T . When the balance table is refreshed, at this moment, if a job arrives at the cloud partition, it will bring the inconsistent problem. The system status will have changed but the information will still be old. This may lead to an erroneous load strategy choice and an erroneous nodes order. To resolve this problem, two Load Status Tables should be created as: Load Status Table 1 and Load Status Table 2. A flag is also assigned to each table to indicate Read or Write. When the flag = "Read", then the Round Robin based on the load degree evaluation algorithm is using this table. When the flag = "Write", the table is being refreshed, new information is written into this table. Thus, at each moment, one table gives the correct node locations in the queue for the improved Round Robin algorithm, while the other is being prepared with the updated information. Once the data is refreshed, the table flag is changed to "Read" and the other table's flag is changed to "Write". The two tables then alternate to solve the inconsistency. The process is shown in Fig.4.

5.3 Load balancing strategy for the normal status

When the cloud partition is normal, jobs are arriving much faster than in the idle state and the situation is far more complex, so a different strategy is used for the load balancing. Each user wants his jobs completed in the shortest time, so the public cloud needs a method that can complete the jobs of all users with reasonable response time. Penmatsa and Chronopoulos proposed a static load balancing strategy based on game theory for distributed systems. And this work provides us with a new review of the load balance problem in the cloud environment. As an implementation of distributed system, the load balancing in the cloud computing environment can be viewed as a game. Game theory has non-cooperative games and cooperative games. In cooperative games, the decision makers eventually come to an agreement which is called a binding agreement. Each decision maker decides by comparing notes with each others. In non-cooperative games, each decision maker makes decisions only for his own benefit. The system then reachs the Nash equilibrium, where each decision maker makes the optimized decision. The Nash equilibrium is when each player in the game has chosen a strategy and no player can benefit by changing his or her strategy while the other players strategies remain unchanged.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

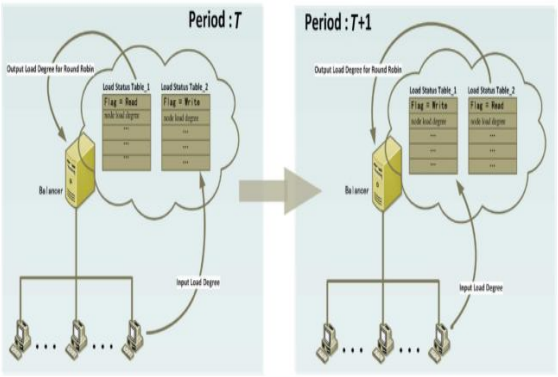


Fig. 4 The solution of inconsistently problem.

There have been many studies in using game theory for the load balancing. Grosu et al.proposed a load balancing strategy based on game theory for the distributed systems as a non-cooperative game using the distributed structure. They compared this algorithm with other traditional methods to show that their algorithm was less complexity with better performance. Aote and Kharat gave a energetic load balancing model based on game theory. This model is related on the dynamic load status of the system with the users being the decision makers in a non-cooperative game. Since the grid computing and cloud computing environment are also distributed system, these algorithms can also be used in grid computing and cloud computing environments. Previous studies have shown that the load balancing strategy for a cloud partition in the normal load status can be viewed as a non supportive game, as described here. The players in the game are the nodes and the jobs. Suppose there are n nodes in the current cloud partition with N jobs arriving, then define the following parameters:

μ_i: Processing ability of each node, $i = 1, \cdots, n$.

ϕ_j: Time spending of each job.

$\Phi = \sum_{j=1}^{N} \phi_j$: Time spent by the entire cloud partition, $\Phi < \sum_{i=1}^{n} \mu_i$.

s_{ji}: Fraction of job j that assigned to node i ($\sum_{i=1}^{n} s_{ji} = 1$ and $0 \leqslant s_{ji} \leqslant 1$).

In this model, the most important step is finding the appropriate value of s_{ji}. The current model uses the method of Grosu et al. called "the best reply" to calculate s_{ji} of each node, with a greedy algorithm then used to calculate s_{ji} for all nodes. This procedure gives the Nash equilibrium to minimize the response time of each job. The strategy then changes as the node's statuses change.

VI. CONCLUSION

Since this work is just a conceptual framework, more work is needed to implement the framework and resolve new problems. Some important points are:

(1) Cloud division rules: Cloud division is not a simple problem. Thus, the framework will need a complete cloud division methodology. For example, nodes in a cluster may be far from other nodes or there will be some clusters in the same geographic area that are still far apart. The division rule should simply be based on the geographic location (province or state).

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

(2) How to set the refresh period: In the data statistics analysis, the main controller and the cloud partition balancers need to refresh the information at a fixed period. If the period is too short, the high incidence will influence the system performance. If the period is too long, the information will be too old to make good decision. Thus, tests and statistical tools are needed to set a reasonable refresh periods.

(3) A better load status evaluation: A good algorithm is needed to set $\text{Load\_degree}_{\text{high}}$ and $\text{Load\_degree}_{\text{low}}$, and the evaluation mechanism needs to be more comprehensive.

(4) Find other load balance strategy: Other load balance strategies may provide better results, so tests are needed to compare different strategies. Many tests are needed to guarantee system availability and efficiency.

REFERENCES

[1] R. Hunter, The why of cloud, http://www.gartner.com/DisplayDocument?doc cd=226469&ref= g noreg, 2012.
[2] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, Cloud computing: Distributed internet computing for IT and scientific research, Internet Computing, vol.13, no.5, pp.10-13, Sept.-Oct. 2009.
[3] P. Mell and T. Grance, The NIST definition of cloud computing, http://csrc.nist.gov/publications/nistpubs/800145/SP800-145.pdf, 2012.
[4] Microsoft Academic Research, Cloud computing,http://libra.msra.cn/Keyword/6051/cloudcomputing?querycloud%20computing, 2012.
[5] Google Trends, Cloud computing, http://www.google.com/trends/explore#q=cloud%20computing, 2012.
[6] N. G. Shivaratri, P. Krueger, and M. Singhal, Load distributing for locally distributed systems, Computer, vol. 25, no. 12, pp. 33-44, Dec. 1992.
[7] B. Adler, Load balancing in the cloud: Tools, tips and techniques, http://www.rightscale. com/info center/whitepapers/Load-Balancing-in-the Cloud.pdf, 2012
[8] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, and C. Mcdermid, Availability and load balancing in cloud computing, presented at the 2011 International Conference on Computer and Software Modeling, Singapore, 2011.
[9] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, N. Nitin, and R. Rastogi, Load balancing of nodes in cloud using ant colony optimization, in Proc. 14th
International Conference on Computer Modelling and Simulation (UKSim), Cambridgeshire, United Kingdom,
Mar. 2012, pp. 28-30.
[10] M. Randles, D. Lamb, and A. Taleb-Bendiab, A comparative study into distributed load balancing algorithms for cloud computing, in Proc. IEEE 24th International Conference on Advanced Information Networking and Applications, Perth, Australia, 2010, pp. 551-556.
[11] A. Rouse, Public cloud, http://searchcloudcomputing.techtarget.com/definition/public-cloud, 2012.
[12] D. MacVittie, Intro to load balancing for developers The algorithms, https://devcentral.f5.com/blogs/us/introtoload-balancing-for-developers-ndash-the-algorithms,2012.
[13] S. Penmatsa and A. T. Chronopoulos, Game-theoretic static load balancing for distributed systems, Journalof Parallel and Distributed Computing, vol. 71, no. 4, pp. 537-555, Apr. 2011.
[14] D. Grosu, A. T. Chronopoulos, and M. Y. Leung, Load
balancing in distributed systems: An approach using cooperative games, in Proc. 16th IEEE Intl. Parallel and Distributed Processing Symp., Florida, USA, Apr. 2002,
pp. 52-61.
[15] S. Aote and M. U. Kharat, A game-theoretic model for dynamic load balancing in distributed systems, in Proc. The International Conference on Advances in Computing, Communication and Control (ICAC3 '09), New York, USA, 2009, pp. 235-238.

BIOGRAPHY

B.SREEKANTH, M.Tech (Department of Computer Science Engineering)VEMU Institute of Technology, chittoor(Dist), A.P, India

G.LOKESH, Associate professor ASSISTANT PROFESSOR (Department of Computer Science Engineering) VEMU Institute of Technology, chittoor(Dist), A.P, India.