International Journal of Grid Distribution Computing
Vol. 8, No. 3, (2015), pp.167-178
http://dx.doi.org/10.14257/ijgdc.2015.8.3.17

Load Balancing with Tree Parity Machine

Ranjan Kumar Mondal, Rajesh Bose and Debabrata Sarddar

Ph.D. Student, Department of Computer Science & Engineering,
University of Kalyani, Kalyani, India
E-mail - ranjan@klyuniv.ac.in
Senior Project Engineer, Simplex Infrastructures Ltd.
Kolkata, India
E-mail - bose.raj00028@gmail.com
Assistant Professor, Department of Computer Science & Engineering,
University of Kalyani, Kalyani, India
E-mail –dsarddar1@gmail.com

Abstract

With rapidly growing influence and demand for cloud computing services, issues involving control, migration, security, availability of data, trust matters directly related to sensitive information stored can no longer be treated lightly. Through our work as explained in this paper, we discuss these topics, and also review several algorithms which are used in cloud computing. These algorithms are implemented in a cloud computing architecture to balance load to improve efficiency. Weighted active monitoring load balancing algorithm, dynamic load balancing algorithm, static algorithms, and ant colony algorithm are all used for load balancing in distributed computing systems. In this paper, we propose a load balancing technique based on Tree Parity Machine. Our approach is to design a Tree Parity Machine based model that would distribute workload evenly among all nodes. The technique proposed by us is not complicated. It has been conceived such that it can work efficiently with training sets that have been proven to produce intended results. Our Tree Parity Machine based model would predict the demand and, accordingly, allocate necessary resources. Thus, it would be able to maintain active servers to meet existing demands. As a result, consumption of energy can be significantly lowered. The conservative approach of over-provisioning is, thus, avoided.

Keywords*: Load balancing, Tree parity machine, Dynamic load balancing, ANN, Cloud computing*

1. Introduction

Cloud computing [1] has been defined as a distributed and parallel architecture comprised of virtual computers interconnected with each other. These, while being provisioned at runtime, are presented to a consumer as combined resources subject to service level agreements entered by and between a cloud service provider and the consumer(s). Although cloud computing has established itself firmly in the market today, it continues to evolve and present a host of challenges. Some of these challenges have been enumerated below:

1. Ensuring that proper access controls are maintained.
2. Network level migrations for ensuring that jobs require least amount of time and money to be moved.
3. To ensure that an effective security system be in place for data either in transit or at rest.

ISSN: 2005-4262 IJGDC

4. Solving data availability issues.
5. Legal compliance and trust issues.
6. Prevent inadvertent release of sensitive information to unauthorized entities.
7. Load balancing.

The last item in the list is by no means the least significant of all the problems which face administrators and users alike of cloud computing. In the course of load balancing, several key data are exchange among the computer hardware powering a cloud system. The numbers generated from processing rates, job arrival and queue rates are closely needed to be monitored. To handle these parameters, several algorithms have been designed and implemented. In our paper, we have tried to examine some of these algorithms against a set of uniform criteria. Through our work, we have tried to analyze and solve the problems related to availability and performance among other factors. These are factors which arise out of improper load balancing.

Our paper is composed of several sections. In Section 2, we discuss load balancing in cloud computing environment. We also introduce types of load balancing and the different policies enforced by dynamic load balancing. In Section 3, we discuss genetic algorithm and present a flowchart to explain a cycle of this algorithm. We explain our implementation of Tree Parity Machine algorithm in Section 4. We delve deeper in Section 4.1, wherein we explain Generation of Queries. We explain a TPM model in Section 4.2. In Section 4.3, we explain the protocol behind TPM design. We introduce our proposed Load Balancing Algorithm with Tree Parity Machine in Section 5. Our proposed algorithm is detailed in Section 6. We analyze the results of our work in section 7. In Section 8, we present our conclusions. Finally, we acknowledge, and cite references used for our work in Sections 9 and 10, respectively.

2. Load Balancing in Cloud Computing

Load balancing [2] enables networks and applications to achieve maximum throughput in the minimum amount of time it takes to respond. Data can be channeled with minimum of delay by apportioning network traffic among servers. Algorithms help in distributing traffic between servers. Although there are a host of algorithms, in the absence of an effective load balancing mechanism, users would be forced to face delays, timeouts, and unacceptable levels of high system latency. With the help of load balancing, cloud systems are able to utilize servers which are not under load, or can accommodate tasks in queue. As a result optimum load distribution and enhanced communication levels can be achieved. In turn, web pages become consistently accessible consequently resulting in overall improved user experience.

Load balancers are of two types: There is a cooperative load balancing technique; and a non-cooperative one. In the cooperative load balancing technique, all the nodes act in concert in order to optimize the overall response times. In the non-cooperative load balancing technique, tasks operate independent of each other to shorten the times taken for local tasks to respond.

Load balancing algorithms are usually segregated into two types – static and dynamic. A static load balancing algorithm ignores the preceding condition or behavior of a given node while distributing load. Dynamic load balancing algorithm, on the contrary, checks previous state while processing distribution of load. Dynamic load balancing algorithm can either be applied as a distributed or non-distributed version. Dynamic load balancing offers the advantage of not halting the system even if a node fails. In such an event of a node failure, only the system performance is left affected. Nodes are able to interact more freely and exchange a far great deal of messages in a load balancing system which is dynamic, rather than in static load balanced system. More messages are delivered throughout the network while selecting an appropriate server using dynamic load

balancing system. This is because real-time communication is required with the remaining nodes present in the network. Dynamic load balancers make use of policies to keep track of information that is updated. Four types of policies are used by dynamic load balancers. These are as follows:

- Transfer policy – this is used in situations where a job is selected and is required to be transferred from a local node to a remote on.
- Selection policy – this used whenever processors carry out information exchange.
- Location policy – this is used where destination node is to be selected to make the transfer.
- Information policy – this is used to record all the nodes in the system.

The job of load balancing is collectively assigned among distributed nodes. Dynamic load balancing can be achieved in two distinct ways within a distributed system. In cases where distributed architecture is in place, all the nodes present execute dynamic load balancing algorithm and jointly share the task of scheduling. In case of an undistributed system, nodes do not share such. Each node works independent of the other while trying to achieve the same aim. We will be demonstrating the use of three different factors in order to compare load balancing algorithms:

- Throughput – By computing the total number of jobs processed within a set duration, not taking into account virtual machine creation and destruction times, throughput figures can be estimated.
- Execution time – This can be figured by computing the times taken for completion of formation of virtual machines, time it takes to process a job, and time taken for destruction of a virtual machine.

Almost all cloud computing algorithm adheres to the following steps to make decisions while processing jobs:

- A request is made by a client. The request is taken up by a cloud coordinator.
- The task is divided into cloudlets by the cloud coordinator. The cloudlets are sent to data centers.
- Coordinators of data centers work on task scheduling. Selection of algorithm to be used for task scheduling takes place.

3. Genetic Algorithm

A Genetic Algorithm (GA) [3] operates on a problem to locate its solution from a list of possible solution. GA is based on a population probabilistic technique. To give rise to a new generation, individuals of a population exchange genotypes in the algorithm evolution and in accordance with their fitness values and given set of probabilistic transition rules. The new generation of individuals is evaluated on the basis of individual fitness functions provided by programmers. Higher the fitness, greater are the chances of being selected. The individuals are allowed to “reproduce” to bring forth one or more than one offspring. Subsequently, the offspring are mutated at random. The process continues till a suitable solution is found, or until a certain number of generations have been produced and passed depending on the programmer. In this paper, this algorithm has been used to produce input weights for the Tree Parity Machine.

The following steps encapsulate the generation of optimum weights using genetic algorithm. These are: initialization, selection, reproduction, and termination. These have been illustrated in Figure 1. A set of random numbers in the range [-L, L] is taken at the start as the first population weight vector. An assumption is made that the fitness function f(x) is parabolic and may, hereby, be defined as,

$$F(x) = \begin{cases} -L & x < -L \\ x^2 & -L \le x < L \\ L & x \ge L \end{cases}$$

For each string in the population, the fitness value is computed. Subject to the fitness value, the most fitted strings from the population are identified using the Roulette Wheel selection method. In this form of selection procedure, the probability of being selected in direct proportion to the respective fitness values, are assigned to individuals. An offspring is produced using two individuals chosen randomly on the basis of the probabilities assigned. New weights are generated on the selected string crossover, and the values are mutated based on the crossover rate (Pc) and mutation rate (Pm). These weights, in turn, are considered as inputs for subsequent run of the algorithm. This completes a single cycle of a GA process. If the condition for termination is encountered, the iterations are stopped. The new population generated would be considered to be as the optimal solution. The GA cycle is illustrated in Figure 1.

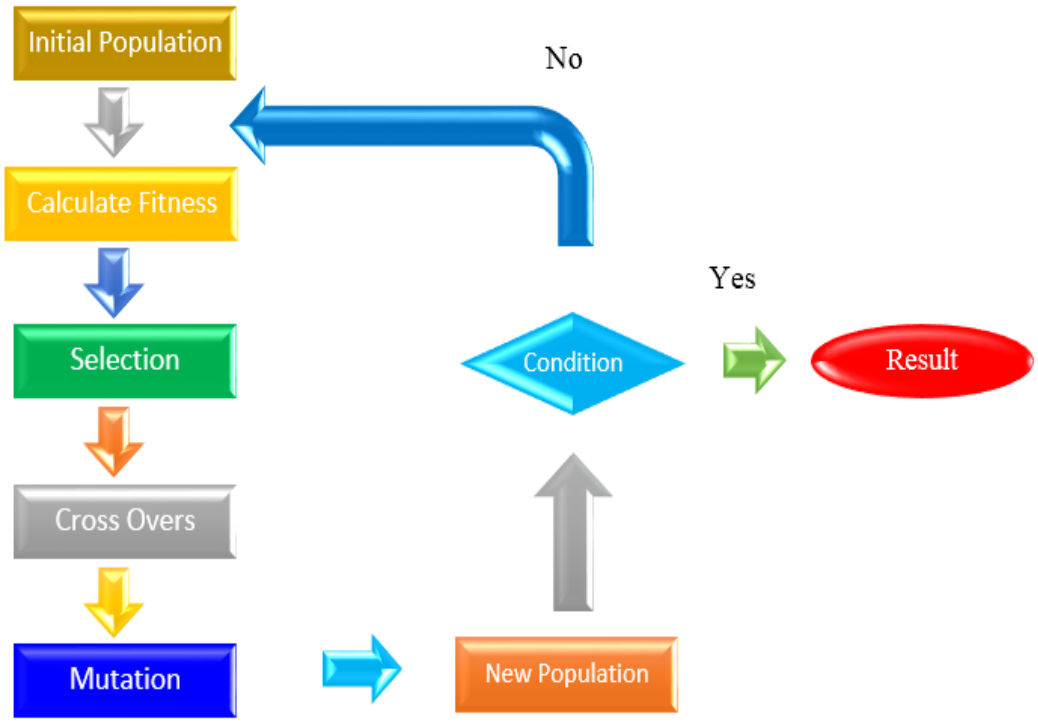

Figure 1. GA Cycle

4. Tree Parity Machine Implementation

The Tree Parity Machine [4] is synchronized through use of genetic algorithm. This is shown in Figure 2. The synchronization is made possible in the following manner as described:

- Weight values obtained from GA process in the range [-L,L] is initialized.

- The following steps are executed until a full synchronization is achieved:
  - A random input vendor is generated.
  - The values of the hidden neurons are calculated.
  - The output neuron value τ is computed.
  - Compare the values τ of both Tree Parity Machines
    - In case outputs differ, go back to step 1.
    - In case outputs are identical, any one of the following learning rules is applied to the weights.

Only the hidden unit σ_i identical to τ changes its weight, as shown in this example. In case any component Wi is pushed out of the interval [-L,+L] by this training step, it is replaced by either +L or –L in a corresponding manner.

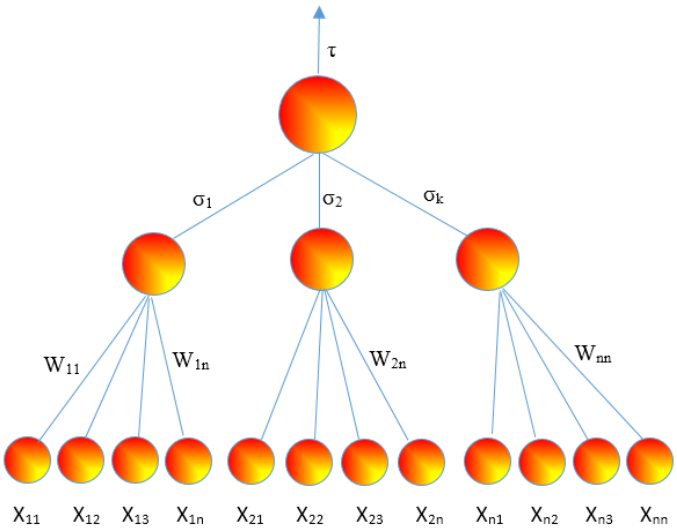

Figure 2. Tree Parity Machine

Upon acquiring full synchronization, the weights of both the Tree Parity Machines are found same. These weights are then used as keys by networks A and B.

4.1. Generation of Queries

Local field of weight vectors are used to generate an input vector. This, in turn, forms the basis of a query. Queries are exchanged as they get generated alternatively by A & B. The public input produced by a pseudo random generator is replaced by the query, as applied in basic neural cryptography algorithm [5]. The process of synchronization is now dependent not only on the synaptic department of TPM, but also on queries as a result of inclusion of the query. The procedure for query generation is described as follows:

Considering that both weights $w_{k,j}$ and inputs $x_{k,j}$are discrete, there can be only 2L+1 possibilities for $w_{k,j} \cdot x_{k,}$. Hence, the solution may be described by counting the number of products with $w_{k,j} \cdot x_{k,j=1}$. The local field is then given by:

$$h_k = \frac{1}{\sqrt{N}} \sum_{l=1}^{L} (c_{k,l} - c_{k,-l})$$

In order to produce our queries, we use the following algorithm in our simulations. The output of σk of the hidden input is first chosen randomly. The set value, therefore, of the local field is given by hk=σkH. We then use either

$$c_{k,l} = \left\lfloor \frac{n_{k,l}+1}{2} + \frac{1}{2l}\left(\sigma_k H\sqrt{N} - \sum_{j=l+1}^{L} j(2c_{k,j} - n_{k,j}) \right) \right\rfloor \quad (3)$$

or

$$c_{k,l} = \left\lceil \frac{n_{k,l}-1}{2} + \frac{1}{2l}\left(\sigma_k H\sqrt{N} - \sum_{j=l+1}^{L} j(2c_{k,j} - n_{k,j}) \right) \right\rceil \quad (4)$$

to arrive at the values of $c_{k,L}$, $c_{k,L-1}$, . . . , ck,1. To obviate chances of rounding errors influencing the average result, one of two equations in each calculation is selected at random with equal probability. In addition to the above, we also take into consideration that $0 \leq c_{k,l} \leq n_{k,l}$. Hence, $c_{k,l}$ is set to the nearest boundary value in case (3) or (4) yields a result beyond this range. The input vector x_k is generated thereafter. As they do not influence local fields, inputs related to zero weights get chosen at random. The input bits $x_{k,j}$ are segregated into L groups in accordance with the absolute value $l = |w_{k,j}|$ of their corresponding weights. In each of the groups, $c_{k,l}$inputs are selected randomly and are set to $x_{k,j} = \mathrm{sgn}(w_{kj})$. The remaining $n_{k,l} - c_{k,l}$input bits are set to $x_{k,j} = -\mathrm{sgn}(w_{k,j})$.

4.2. The TPM Model

The Tree Parity Machine is a unique and a special form of multi-layer feed-forward neural network. It consists of three types of neurons. These consist of one output neuron, “K” hidden neurons and “KxN” input neurons. Three values are passed as inputs to the network:

$$x_{ij} \in \{-1, 0, +1\}$$

The weights between input and hidden neurons receive the values as determined by

$$w_{ij} \in \{-L, ..., 0, ..., +L\}$$

The following formula is used to calculate output value of each hidden neuron. This is computed as a sum of the products of input neurons and the weights.

$$\sigma_i = \mathrm{sgn}\left(\sum_{j=1}^{N} w_{ij} x_{ij}\right)$$

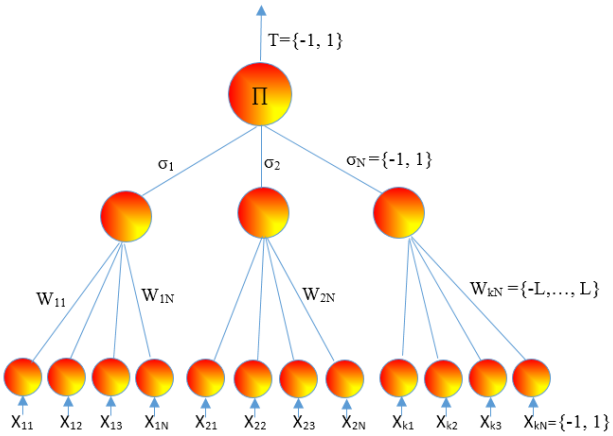

Figure 3. Tree parity Machine with L=[-4,4], K=3 and N=4

The following formula, Signum, is a simple function which returns -1, 0 or 1:

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

In order to ensure that a binary output value is produced, the output of the hidden neuron is mapped to -1 in cases where the scalar product is 0. The output of neural network is then calculated as the product of all values produced by hidden elements.

$$\tau = \prod_{i=1}^{K} \sigma_i$$

The output of Tree Parity Machine is binary.

4.3. Protocol

Each of the two parties (A and B) uses its own Tree Parity Machine. The Tree Parity Machines are synchronized by going through the steps as given under:

1. Set random weight values.
2. Execute the following steps till full synchronization is achieved:
    1. Generate a random input vector X.
    2. Calculate the values of hidden neurons.
    3. Compute the value of the output neuron.
    4. Compare the values of both the Tree Parity Machines
        1. If outputs are different, then go back to step 2.1.
        2. In case outputs are same, one of the suitable learning rules is applied to the weights.

Upon achieving full synchronization, i.e., when the weights w_{ij} of both Tree Parity Machines are identical, A and B can begin using weights as keys. This is known as a bidirectional learning method. Any of the following learning rules can be used for synchronization:

- Hebbian learning rule:
$\omega_i^+ = \omega_i + \sigma_i x_i \Theta(\sigma_i \tau) \Theta(\tau^A \tau^B)$

- Anti-Hebbian learning rule:
$\omega_i^+ = \omega_i - \sigma_i x_i \Theta(\sigma_i \tau) \Theta(\tau^A \tau^B)$

- Random walk:
$\omega_i^+ = \omega_i + x_i \Theta(\sigma_i \tau) \Theta(\tau^A \tau^B)$

5. The Proposed Load Balancing Algorithm with TPM

We propose to introduce a device in cloud systems that would examine the current performance profiles of all available cloud resource nodes. Information collected by our proposed device would be fed into TPM load balancer. The TPM load balancer would be finally responsible for managing the different states of the cloud resource nodes.

We also propose that cloud computing would have a number of different servers which would collectively analyze the existing performance of all available cloud resource nodes. The architecture of the framework is shown in Figure 1.

Our proposed Tree Parity Machine has been built keeping in mind simplicity and ease of understanding. It functions using prediction ideas. Our proposed TPM is composed of three layers. The first layer is the input layer. This represents the current workload for a given node. The second layer is the hidden layer. The third is the output layer which stands for the balanced workload for N nodes. Each node in the input layer represents either the workload of the current server or the average workload of a cluster of servers. An integral number is assigned to each node in the input layer. The corresponding node in the output layer, on the other hand, denotes either the workload of server or the cluster's average workload after balancing. Therefore, there are an equal number of TPMs in both the input and output layers. The process of load balancing is accomplished through training of TPM on various and illustrative examples of balanced and unbalanced instances. It is apparent that a major reduction in energy consumption levels can be achieved by selectively and systematically shutting down servers or switching these to sleep mode when not in use.

6. Algorithm

Step 1: The minimal execution time service node is chosen from C service nodes based on the requirement of subtasks Xi. A Min-time service node is formed where Xi is the total number of subtasks, C is the total number of nodes inside an executable subtask service nodes set.

Step 2: To choose service node γ from Min-time set in which γ has the shortest execution time. Here, γ is the identifier number of the service node.

Step 3: Service node γ is assigned a subtask.

Step 4: From the task set that is needed to be executed, remove the completely executed subtask. Where, Xi is represented as the identifier of the subtask.

Step 5: Following rearrangement of the Min-time array, service node γ is to be inserted at the very end.

Step 6: Repeat steps 1 through 5 until all subtasks are executed completely.

7. Result

In order to train Artificial Neural Network [6], actual workloads are applied at the input layer such that the outputs are calculated at the output layer. These can then be compared with desired loads, errors identified, and weights adjusted. The process is allowed to continue and cycle over and over again with large set of examples until the ANN is trained with an accepted error rate. The network is, subsequently, tested with different data set soon after it is trained with tolerable margins of error. If otherwise, the ANN must undergo retraining with more examples and, even possibly, with change in the parameters directly involved in training. Thus, when the ANN is deemed to have sufficiently acquired knowledge, the training is finally brought to an end. To ensure that good quality learning is infused in the ANN, care has to be taken to select relevant and pertinent examples. To further ensure that good performance can be obtained, the number of neurons in each hidden layer, as well as the number of hidden layers, too, is changed during training. A great number of load balancing techniques are employed in distributed and cloud systems. Almost all these rely on methods which target reduction of overhead, improving service response times, and improving performance, etc. But none of these techniques that have been discussed here consider factors such as power consumed and resultant carbon emissions (i.e., matrices 9 and 10). We are of the view that there is a scope for design and development of an energy efficient load balancing system that can offer improvements in terms of cloud computing performance through balancing of

workload across all nodes in a cloud. Such a design would help in maximizing resource utilization and reduce energy consumption and carbon emission levels to an extent where it Green computing is achieved and made possible. Our proposed method makes use of green elements in the design and construction of a load balancer (i.e., ANN). The ANN should be able to predict demand and, accordingly, allocate resources in direct relation and proportion to the demand.

The TPM comprises several interconnected processing units that send signals to each other depending on the signals that each receives. This functionality is similar to artificial neural networks. The three fundamental elements of the TPM are as follows:

7.1. Connecting links

These provide weights, w_j, to the input values x_j for j = 1 ...m;

7.2. Adder

An adder sums the weighted input values to arrive at the input value to be fed to the activation function $v = w_0+m\sum_j^{=1}w_jx_j$; where w_0 is called the bias (entirely different from statistical bias in prediction or estimation) and which is a numerical value. It would facilitate in noting this bias as being the weight for an input x_0 whose value is always equal to 1, and such that $v =m\sum_j^{=0}w_jx_j$;

7.3. Activation Function

An activation function "g" (also referred to as a squashing function) that maps "v" to g (v) to produce the output value of the TPM.

An input vector is applied to a neuron via input connections. The weights of these connections keep on changing as TPM learns. It is possible for the weights to either stimulate or slow down transmission of the input value. Mathematically, input values are multiplied by the value of that particular weight. At the neuron node, the weighted inputs are added up. The summation result is, thereafter, passed to an activation function. Some of the examples of activation functions are: step function, ramp function and sigmoid function.

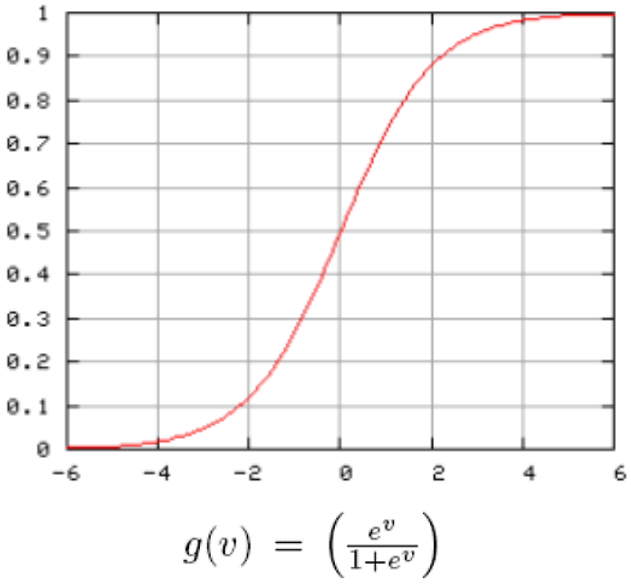

Figure 5. Sigmoid Function

The horizontal axis in Figure 5 represents the sum of the weighted inputs. The output of the function for each value on the horizontal axis is plotted and the resultant curve can

be seen. The function output is, thereafter, fed to each of the nodes that are connected to output weights of the firing neurons.

8. Conclusion

Our proposed technique is a step toward achieving demand prediction. Through our design, we have shown how TPM learning methods can be infused in our model to help deduce load patterns and, thus, enable prior allocation of resources according to estimated demands. Using our proposed design, servers and computing resources can be utilized without the need to overprovision. We have discussed the algorithm of our proposed design. It is our belief that with the aid of TPM and load balancing method that is presented in our work, the existing resources of a cloud system could be put to more efficient use. The overall impact of our proposed model can lead to realizing the aim of Green computing in a cloud environment.

Acknowledgment

We would like to express our gratitude to Dr. Kalyani Mali, Head of Department, Computer Science and Engineering of University of Kalyani. Without her assistance and guidance, we would not have been able to make use of the university's infrastructure and laboratory facilities for conducting our research.

Reference

[1] 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, ISSN: 0167-739X, Elsevier Science, Amsterdam, The Netherlands, vol. 25, no. 6, **(2009)** June pp. 599-616,.
[2] N. A. Singh, M. Hemalatha, "An approach on semi distributed load balancing algorithm for cloud computing systems" International Journal of Computer Applications,vol. 56, no.12, **(2012)**.
[3] K. Deb, *et. al.*, "A fast and elitist multiobjective genetic algorithm, NSGA-II." Evolutionary Computation, IEEE Transactions on, vol. 6, no. 2, **(2002)**, pp. 182-197.
[4] I. Kanter, I, W. Kinzel and E. Kanter, "Secure exchange of information by synchronisation of neural networks", Europhysics Letters, vol. 57, **(2002)**, pp. 141–147
[5] P. Revankar, W. Z. Gandhare and D. Rathod, "Neural Synchronization with queries",Signal Acquisition and Processing, 2010. ICSAP'10. International Conference on. IEEE, **(2010)**.
[6] A.Sallami, M. Nada and S. A. A.Alousi, "Load Balancing with Neural Network",International Journal of Advanced Computer Science and Applications (IJACSA)vol. 4, no.10,**(2013)**.

Authors

Ranjan Kumar Mondal, received his M.Tech in Computer Science and Engineering from University of Kalyani, Kalyani, Nadia; and B.Tech in Computer Science and Engineering from Government College of Engineering and Textile technology, Berhampore, Murshidabad, West Bengal under West Bengal University of Technology, West Bengal, India. At present, he is a Ph.D research scholar in Computer Science and Engineering from University of Kalyani. His research interests include Cloud Computing, Wireless and Mobile Communication Systems.

Rajesh Bose, is currently pursuing Ph. D. from Kalyani University. He is an IT professional employed as Senior Project Engineer with Simplex Infrastructures Limited, Data Center, Kolkata, India. He received his degree in M. Tech. in Mobile Communication and Networking from WBUT in 2007. He received his degree in B.E. in Computer Science and Engineering from BPUT in 2004. He also

has several global certifications under his belt. These are CCNA, CCNP-BCRAN, and CCA (Citrix Certified Administrator for Citrix Access Gateway 9 Enterprise Edition), CCA (Citrix Certified Administrator for Citrix Xen App 5 for Windows Server 2008). His research interests include Cloud Computing, Wireless Communication and Networking.

Debabrata Sarddar, is an Assistant Professor at the Department of Computer Science and Engineering, University of Kalyani, Kalyani, Nadia, West Bengal, India. He completed his PhD from Jadavpur University. He did his M. Tech in Computer Science & Engineering from DAVV, Indore in 2006, and his B.E in Computer Science & Engineering from NIT, Durgapur in 2001. He has published more than 75 research papers in different journals and conferences. His research interests include Wireless and Mobile Systems and WSN, and Cloud computing.