International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 3; Issue: 6; December -2014

AN APPROACH FOR IMPROVING ELASTICITY IN CLOUD COMPUTING USING NEURAL NETWORK APPROACH

Aparna Manikonda [1], Poonam Tijare [2], Shruthi K [3]

[1,2,3] Department of Computer Science and Engineering, CMRIT, Bangalore, India [1] aparna.m@cmrit.ac.in [3] shruthi.k@cmrit.ac.in
[2] poonam.v@cmrit.ac.in

Abstract **- The main concern for improving elasticity in cloud computing is how to handle the scheduling of tasks and balancing the load in cloud network with high throughput, less over head and response time there by efficient utilization of resources. As a result, Dynamic approaches with the purpose of scaling the tasks and resources efficiently have drawn attentions of many research studies. Recently, there have been a strong interest to use intelligent tools especially Neural Networks in improving elasticity of cloud infrastructure , due to their simple parallel distributed computation, distributed storage, data robustness, auto-classification of tasks and load in cloud centers . Dimensionality reduction and prediction of cloud data obtained simply from the outputs of the neural-networks algorithms can lead to lower communication costs and energy conservation. This paper aims to present the most important possible application of neural networks for improving elasticity in cloud computing.**

Keywords— Cloud computing, SOM, Neural Networks, Task Scheduling, Resource allocation, Load Balancing.

1.Introduction

Organizations are tired of mobilizing & committing large cap-ex that tie up cash and require also annual maintenance payments. However, with the introduction of cloud computing, people as well as business enterprises can now access their programs through the internet. This kind of computing is rapidly growing in popularity and especially with small business enterprises. As the number of users on cloud increases, the existing resources decreases automatically which leads to the problem of delay between the users and the cloud service providers. Thus, improving the elasticity features ***(Load balancing and Task Scheduling)*** in cloud there by dealt smartly with traffic such that the situation in which some nodes are overloaded and some other are under loaded should never arise.

So many research studies focused on improving the elasiticity features in cloud which are based on different considerations like static, dynamic, distributed, centralized, hierarchical and workflow dependent approaches[1-4]. In static approach Prior knowledge base is required about, each node statistics and user requirements are not compatible with changing user requirements as well as load. In Dynamic approach, Run time statistics of each node are monitored to adapt to changing load requirements but is time consuming. Where as in Centralized and Distributed has a different approach together. In the former case, Single node or server is responsible for maintaining the statistics of entire network and updating it from time to time where as in latter case all the processors in the network responsible for load balancing store their own local database (e.g. MIB) to make efficient balancing decisions. But in both these approaches the overhead of computation is more. Hierarchical Nodes at different levels of hierarchy communicate with the nodes below them to get information about the network performance and it has a very complex approach[5-6].

In this paper we present a novel algorithm for improving the elasticity in cloud through using of Self organizing map neural networks is presented which can provide a uniform distribution of load in all data centers. The difference between our proposed approaches with previous approach is that it is able to adaptively select next data center not only based on their topological closeness (distances) but also based on their load and their density in each set-up phase by using SOM neural network. We tried to develop the classic idea for load balancing and task scheduling and incorporate a data center allocation model using SOM neural networks in order to apply three unrelated variables.

2. KOHONEN SELF ORGANIZING MAP

The Kohonen self-organising networks[7] have a two-layer topology. The first layer is the input layer, the second layer is itself a network in a plane. Every unit in the input layer is connected to all the nodes in the grid in the second layer. Furthermore the units in the grid function as the output nodes. The nodes in the grid are only sparsely connected. Here each node has four immediate neighbours.

The network (the units in the grid) is initialised with small random values. A neighbourhood radius is set to a large value. The input is presented and the Euclidean distance between the input and each output node is calculated. The node with the minimum distance is selected, and this node, together with its neighbours within the neighbourhood radius, will have their weights modified to increase similarity to the input. The neighbourhood radius decreases over time to let areas of the network be specialised to a pattern. .The algorithm results in a network where groups of nodes respond to each class thus creating a map of the found classes.

2.1. Algorithm of Kohonen Self Organizing Map

The Self-Organizing Map algorithm can be broken up into 6 steps [7]

STEP-1 Each node's weights are initialized.

STEP-2 A vector is chosen at random from the set of training data and presented to the network.

STEP-3 Every node in the network is examined to calculate which ones' weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).

STEP-4 The radius of the neighborhood of the BMU is calculated. This value starts large. Typically it is set to be the radius of the network, diminishing each time-step.

STEP-5 Any nodes found within the radius of the BMU, calculated in 4), are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its' weights are altered.

STEP-6 Repeat step-2 for N iterations.

3. PROPOSED APPROACH

3.1. FRAMEWORK

The operation of the algorithm is divided in two phases. Each phase begins with a data center selection phase, in which the parameters for a particular data center is calculated, followed by a allocator phase, where the data center will be allocated to the specified machine. Data center allocator aggregates the request received and allocates the data center to the specified end user request.

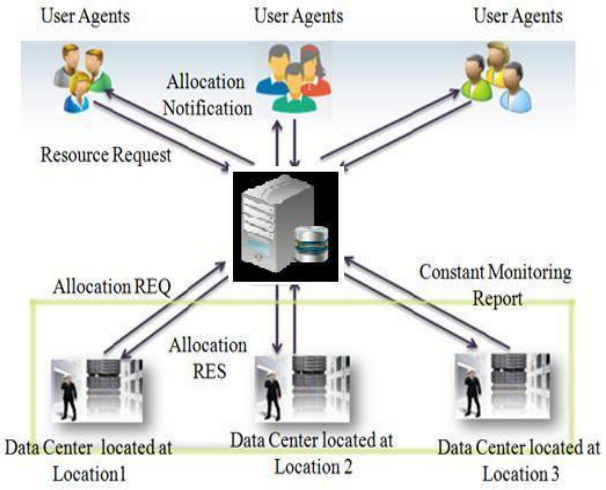

Figure.1. Frame Work of the proposed Approach

3.2. Data center Selection phase/ Learning Phase

In this phase, neurons from the second layer compete for the privilege of learning among each other, while the correct answer(s) is (are) not known. This implies that for a certain input vector, there is only one neuron that gets activated. To determine which neuron is going to be activated, the input vector is compared with the vector that is stored in each of the neurons, the so-called synaptic weight-vectors. Only the neuron whose vector most closely resembles the current input vector dominates. Consequently, the weights of the winning neuron and its neighbouring neurons are updated by a neighbourhood function.

In order to organize the neurons in a two dimensional map, we need a set of input samples

X(t)=[Distance-D(t), Workload- W(t), Power usage effectiveness-P(t), estimated allocation delay time-E(t)]

These are the set of variables that we want to consider as SOM input dataset; these samples should consider all the QoS environments in which a communication link between a pair of data center that can work.

So we will have a D matrix with n*4 dimensions. Since we are applying two different type variables, first we have to normalize all values. We used a Min-Max normalization method [] in which min_a and max_a are the minimum and maximum values for attribute a. Min-max normalization, maps a value v in the range of (0, 1) by simply computing.

$$V' = \frac{(v - \min_a)}{(\max_a - \min_a)} \quad (1)$$

By means of above equation, the dataset D matrix will be:

$$D = \begin{matrix} \frac{D_1}{D_{MAX}} & \frac{W_1}{W_{MAX}} & \frac{P_1}{P_{MAX}} & \frac{E_1}{E_{MAX}} \\ \vdots & \vdots & & \vdots \\ \frac{D_n}{D_{MAX}} & \frac{W_n}{W_{MAX}} & \frac{P_n}{P_{MAX}} & \frac{E_n}{E_{MAX}} \end{matrix} \quad (2)$$

In order to determine weight matrix, BS has to select m cluster heads nodes corresponding to m regions of the network space. We need four variables of these selected data center to apply them as weight vectors of our SOM: Geographical distance between consumer and data center {D(t)}, workload of each data center{W(t)} , power usage effectiveness{P(t)}, estimated allocation delay time{E(t)}. Therefore our weight matrix would be:

$$W = \begin{matrix} \frac{D_1}{D_{MAX}} & \cdots & \cdots & \frac{D_n}{D_{MAX}} \\ \frac{W_1}{W_{MAX}} & \cdots & \cdots & \frac{W_n}{W_{MAX}} \\ \frac{P_1}{P_{MAX}} & & & \frac{P_n}{P_{MAX}} \\ \frac{E_1}{E_{MAX}} & \cdots & \cdots & \frac{E_n}{E_{MAX}} \end{matrix} \quad (3)$$

Where W is the weight matrix of SOM, we have a 4*m weight matrix. The SOM topology structure would be as Fig.2.

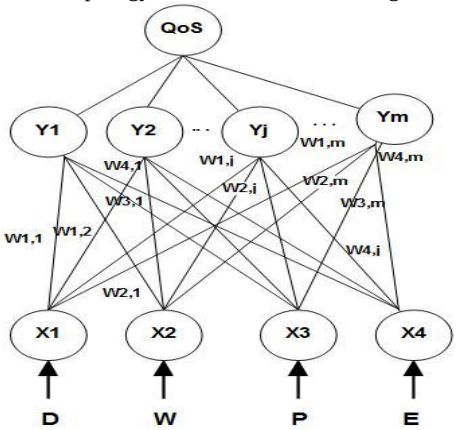

Figure 2. SOM topology structure

In our application, learning is done by minimization of Euclidian distance between input samples and the map prototypes weighted by a neighbourhood function $h_{(i,j)}$:

$$E_{SOM} = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{M} h_{i,N}\, x^{(k)} \left\| W_j - x^{(k)} \right\|^2 \quad (4)$$

Where N is the number of data samples, M is the number of map units; $N(x^{(k)})$is the neuron having the closest referent to data sample $x^{(k)}$ and h is the Gaussian neighborhood function defined by:

$$h_{i,j}(t) = exp\left(-\frac{\|r_j - r_i\|^2}{2\sigma_t^2}\right) \quad (5)$$

where $\|r_i - r_i\|^2$ distance between map unit _j' and _i/p' sample i and is the neighbourhood radius at time t which is defined by

$$\sigma(t) = \sigma_0 \exp\left(-t/T\right) \quad (6)$$

Where *t* is the number of iteration, *T* is the maximum number of iteration or the training length. The distance between X_k^k and weight vectors of all map neurons are computed. A neuron $N(X^k)$ which has the minimum distance with input sample $X^{\cdot}$, would win the competition phase:

$$N(X_k) = arg_{i<j\leq m}\, min \|w_j - X_k\|^2 \quad (7)$$

The neighbourhood radius is a large value at the beginning and it will reduce with increasing of the time of the algorithm in iteration. After competition phase, SOM should update the weight vector of the winner $N(X_k)$ and all its neighbors which placed at the neighborhood radius of ($R^{N(X_k)}$). If $W_j \in (R^{N(X_k)})$then :

$$w_j(t+1) = w_j(t) + \alpha(t) h_{j,N(X_k)} \left(x(t) - w_j(t)\right) \quad (8)$$

Else

$$w_j(t+1) = w_j(t) \quad (9)$$

Where α_0 the initial learning rate, *t* is the number of iteration and *T* is the maximum training length. The learning phase repeats until stabilization of weight vectors.

3.3. Allocator Phase/ Execution Phase

In this phase the weights are declared fixed. First, every neuron (i, j) calculates the similarity between the input vector x(t), $\{x_k|1<k< m\}$ and its own synaptic weight vector W'_{ij}. This function of similarity is based on a predefined similarity criterion. Next, it is declared a winning neuron, with a synaptic weight vector W'_g, similar to the input x. Every Data center implements a SOM as a function. SOM gives an output denoted by QoS. This value is returned by a function defined by the SOM user, according to its aims.

In the execution phase, the data centers operate according to the framework designed as above (fig.1.). Every data center measures the QoS periodically with every neighbor data center, which determines an input sample. After a data center has collected a set of input samples from member nodes, it runs the wining neuron election algorithm. After the winning neuron is elected, the data center uses the output function to assign QoS estimation. Finally, this value is employed to select the request for next data center.

As a consequence of the learning phase, we have declared an output function that has to be run in every data center node. This procedure is named the wining neuron election algorithm.

3.4. Parameters Modelling

3.4.1 Geographical distance between Consumer and data centers (D). It can be calculated as follows, let _u' and _m_i' be the location of consumer and the location of data center _i'.

Let δ be the network delay weight to travel the request message along the path between consumer and data center. The propagation delay time d_{ui} is described as follows:

$$d_{ui} = |u - m_i| \times \delta \quad (10)$$

3.4.2 Workload on data centers(W), this can be calculated as follows, where p_i be a total available number of physical CPU threads in the data center _i'. The VMM considers that the virtual CPU of VMs is in form of logical CPUs, it can be allocated to each corresponding threads of physical CPUs in the data center.

$$w_i = \frac{\sum_{j=1}^{n} \alpha_{ij}}{p_i} \times \Gamma \quad (11)$$

3.4.3 Power usage effectiveness (P) is a measure of how efficiently a computer data center uses its power; specifically, how much of the power is actually used by the computing equipment. The PUE at each of the data center, is calculated by using the formula as follows

$$p = \frac{Total\ Facility\ Power}{Total\ Equipment\ Power} \quad (12)$$

3.4.4 Estimate the allocation delay time (E) To evaluate estimated allocation delay time (E), both workload w_i of data center _i' and geographical distance d_{ui} are summed.

$$E_i = w_i + d_{ui} \quad (13)$$

4.Conclusions

We have presented a new neural network based algorithm for improving elasticity in cloud. The algorithm elects the data center nodes for next incoming request running an AI-algorithm. The use of AI in every data center node dynamically varies the assignment of this node role, distributing the energy consumption through the network. In fact the use of a SOM on every data center node implies the distribution of the artificial intelligence over the network.

References

[1] Zhang, Q., Cheng, L. & Boutaba, R. (2010). Cloudcomputing: State-of-the-art and research challenges.*Journal of Internet Services and Applications*, 1(1), 7-18. DOI 10.1007/s13174-010-0007-6.

[2] Sotomayor, B., Montero, R. S., Llorente, I. M. &Foster, I. (2009). *Virtual infrastructure management in private and hybrid clouds*. IEEE InternetComputing, 13(5), 14-22.

[3] Al Nuaimi, K., Mohamed, N., Al Nuaimi, M. & Al-Jaroodi, J. (2012). *A survey of load balancing in cloud computing: challenges and algorithms*. 2012 IEEE Second Symposium on Network Cloud Computing and Applications 978-0-7695-4943-9/12 College of Information Technology, UAEU Al Ain, United Arab Emirates

[4] Zhao, C., Zhang, S., Liu, Q., Xie, J. & Hu, J. (2009). Independent Tasks Scheduling Based on Genetic Algorithm in Cloud Computing.

[5] M. Randles, D. Lamb, and A. Taleb-Bendiab, ―A comparative study into distributed load balancing algorithms for cloud computing,‖ in Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, pp. 551–556, IEEE, 2010.

[6] Z. Zeng and V. Bharadwaj, ―A Static Load Balancing Algorithm via Virtual Routing,‖ Proc. Conf. Parallel and Distributed Computing and Systems (PDCS '03), Nov. 2003.

[7] T. Kohonen, The self-organizing map, in: Proceedings of the IEEE, vol. 78, 1990, pp. 1464–1480.