Artificial Neural Network Approach for Load Balancing in Distributed Systems

Saba Khalid[1] Riyazuddin Khan[2] Afsaruddin[3] Jameel Ahmad[4]
[1,2,3,4]Department of Computer Science Engineering
[1,2,3,4]Integral University, Lucknow

Abstract— the pace and accessibility of computation in present days are internet dependent, for calculation arranged all the computing device in a grouped that is known as network. For building of network all computing machines like main frame computer, desktop computer, minicomputer are connected to high speed network, and that networked systems are then referred as parallel system or distributed systems. In centralized systems a single computer acts like a controller that has all the responsibilities namely: jobs creation, job migration, monitoring of overloaded system, monitoring of under loaded systems, migration of jobs etc. However, it is clear that the idea of distributing tasks over several hosts has made the problem of efficient task-resource assignment more difficult. This paper implements load migration from overloaded node to under loaded node with the help of artificial intelligence, further under loaded computing node is also searched[2,9]. There emerges efficiency problem from the fact that it is probable to find idle hosts while there are jobs queuing for execution in other hosts. Consequently it would be better to move from heavily loaded hosts to idle hosts. This and other related efficiency problems are solved via load balancing.

Key words: Under Loaded Systems; Overloaded Systems; Load Migration, Load Generation, Distributed Systems

I. INTRODUCTION

A centralized systems are those system in which the central controller has all the responsibilities and it acts as a bottleneck on the grounds that if central controller fails, the whole systems gets deteriorated, however if we perform switching of task among the no of processing elements it means the switching time is wasted[7]. For avoiding of such circumstances the single jobs is broken into a no of jobs and is distributed to all available computing machines, execution of such activity saves the task switching time and also computation and response time is minimized. In distributed systems, a single computational task is divided into several subtasks and distributed across several computers so that the power (CPU, memory, network, disk) of these computers is accumulated. The most striking feature of distributed systems is the accessibility to increase their overall computational power by adding new nodes. This requirement is raised because of the need to address increasingly growing computation workloads. The growing size of the distributed system eventually results in heterogeneous environments as the newly-added nodes frequently have better computational power. Several efforts are being taken to develop technologies to coalesce the power of Internet-connected systems.

Load balancing is a procedure that adjust the load among all over-loaded calculation machine to an under stacked calculation machine it implies that load can be exchanged starting with one machine then onto the next machine, the typical goal of the load balancing is to minimize the execution time. There can be two scenarios in load balancing: the assignments may communicate with one another or every undertaking execute freely with no correspondence. If the tasks does not correspond with one another, than the target is to minimize the execution time of the autonomous procedures. Furthermore, the target must be to minimize the normal finish times of free assignments by re designing them on hosts so that the normal load times of all hosts are almost the same. The undertakings' revamping must be done to fulfill the accompanying primary goals [3-5].

- All hosts must be kept engaged as far as possible.
- The amount of inter-host communication must be minimized.

The aim of first objective is to make all hosts have the comparable measure of workload. This is done primarily by influencing the load from considerably stacked hosts to sit out of gear or less stacked hosts. The objective of second statement is to diminish the utilization of system resource so that the assignments don't invest their energy sitting tight for messages from alternate undertakings running on distinctive hosts. The solution for this issue is to assemble exceedingly conveying undertakings on a solitary host as much as might reasonably be expected[11].

Our main objective is load balancing and the best matching between the tasks and the resources that fulfill the requirements. Though, examining and finding the best matching of T tasks to P processors is an NP-complete problem [6]. Hence, all algorithms try to find an optimal or near-optimal solution in an efficient way. Generally, load balancing algorithms is comprised of three phases: information collection, decision making and data migration (Figure 1). During the information collection phase, load balancer gathers the information of workload distribution and the state of computing environment and detects whether there is a load imbalance. The decision making phase focuses on calculating an optimal data distribution, while the data migration phase transfers the tasks from overloaded processors to under loaded ones.

Load balancing algorithms Because of its vital role in superior performance distributed computing, the load balancing issue has been intense widely as of late and various load adjustment calculations have been invented for parallel processing. The load balancing calculations can be characterized in several classes as indicated by the strategies and information structures they utilized [3, 4, and 15]:

A. Master/Slave

The master/slave load balancing approach is the least complex load balancing algorithm calculation and is very simple to execute. One of the hosts known as the expert is in charge of keeping up the assignments and transmission them to the slaves[4]. The slave hosts complete their assignments and demand new tasks from the expert host. The master/slave methodology has tremendously appealing

components that make it appropriate for some issues[8]. In several cases expert/slave model is apt for issues in which errands can be performed autonomously and not concurrently by a solitary PC. Moreover, it must be conceivable to send an assignment with little messages so that the message correspondence between the slaves and the expert doesn't bring about a bottleneck.

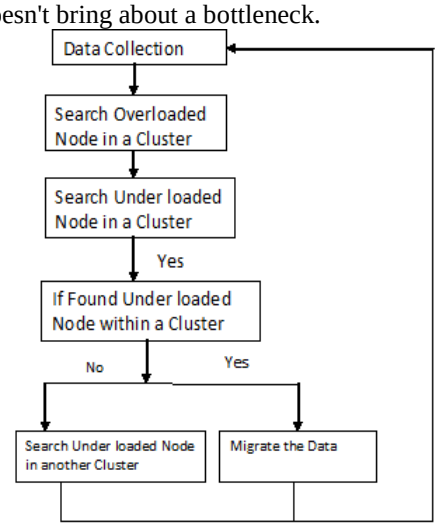

Fig. 1: Load Balancing Phases

II. THE MODEL

In a supposed model, a large number of computing resource are grouped in network, we also initialized large number of task with respect to available computing resource are generated rapidly. Then available computing resource is not sufficient it signify some task should be in a waiting queue, after the mapping of task over the computing resource, remaining task should be in waiting queue if any computing resource is free at that time controller migrate the task to the available computing machine[12]. This model is applicable on heterogeneous task as well as homogeneous task. The state of the system at iteration k is represented by the task distribution vector, w (k), where wi(k) is the number
For processor i,
Wi(k + 1) = wi(k)
- Tasks sent by i computing machine (load balancing)
+ Tasks received by computing machine (load balancing)
+ Tasks arriving during k on a computing machine
- Tasks departing during k computing machine

We represent the transfer of tasks due to load balancing

Suppose Mij (k) be the part of task are allotted to the ith processor after k iteration or kth time. The task distribution at iteration k + 1,is w(k + l), can be denoted as a part of w(k), the task transfer due to load balancing, and the addition and deletion of tasks due to the dynamics of the application. Of its tasks that processor I transfers to processor j[9]. Mij (k) wi (k) therefore is the number of tasks sent by processor i to processor j (j # i). Can now be written as:

$$wi(k + 1) = wi(k) - \Sigma j!=i + Mij(k)wi(k) + \Sigma j!=i\ Mji(k)wj(k) + \lambda i(k) - \mu(k)$$

Where λ (k) and μ (k) are respectively denoting the task arrival and departure rates and represent the dynamically changing workload requirements of the application.

III. LOCAL METHODS

The majority of load balancing calculations use global information to focus on the unmoving hosts and the procedures. That is, all hosts and procedures in the conveyed frameworks are analyzed comprehensively to reduce normal completion time.

Although what might be expected, local load balancing method work in their neighborhood [3, 15]. The point is to spread the workload of exceedingly stacked processors to different processors in the characterized nearby neighborhood. The areas are formed in such a way so as to cover one another in a few cycles and the workload gets spread over all processors. Unlike other global techniques local method technique of neighborhood systems is very quick and reasonable[8,10]. As all the information and communication is performed within small sets of processors, the methods scale well with a number of processors. Local methods are incremental and can be parallelized easily. Single iteration can reduce a single heavily loaded processor's workload notably. Since the total time computation time is determined by the time required by the most heavily loaded processor, a small number of iterations may be sufficient to reduce imbalance [11].

The most critical point of concern of nearby strategies is that it is entirely incremental, quick and proficient since they take a nearest at little neighborhoods. In a couple of cycles, they can lessen the workload of an over-loaded processor. Since the collective calculation time is controlled by the most vigorously stacked processor, a little number of cycles is sufficient to decrease the calculation time to a worthy level. These sorts of calculations are difficult to circulate since they chip away at individual processors and needn't bother with worldwide information. Still when worldwide equalization rather than neighborhood parity is required, a great deal of cycles may be expected to unite to a decent adjust state. Along these lines excellent load adjusting is very troublesome with neighborhood systems. Nearby strategies for the most part are extremely proficient and quick in doing little and neighborhood changes in loaded parity. In dispersed application that does not works concurrently, the neighborhood techniques may be more productive, since worldwide routines are essentially synchronous. At the point when a processor has completed its task, it can start load adjusting to ask for more employments. While different neighborhoods proceed with their assignments, single neighborhoods may do load adjusting. In any case, the drawback here is that offbeat models are hard to actualize. Neighborhood routines have two straightforward steps which exceedingly influence their execution[12,15]. In the first step, the system figures the amount of work that must be moved from the processor to different processors to adjust the heap. In the second step, the strategy chooses the undertakings and processors to be moved. The tasks to be moved are selected by correspondence relations with different job work assignments. This minimization expenses ensures the correspondence's of the tasks[6]. Some sample calculations in this class are spreading, interest driven and dimensional operate.

A. Mesh Partitioning (Graph Partitioning)

The majority important part of load balancing algorithms join graph partitioning algorithms. This type of algorithms model the difficulty domain as a graph consisting of weighted vertices and edges [3, 4]. When we use this model, load balancing difficulty is distorted to a problem with the aim to reduce the imbalance among the nodes and minimize the communication between the nodes. In its most general form, the graph partition tries to find how improved to divide a graph's vertices into a individual number of subsets such that:

1) Equal the number of vertices per subset.
2) Minimizing the number of edges on both sides the subsets A graph is characterize as far as an arrangement of vertices V and an arrangement of edges E. Edges associate vertices from V pair-wise and are undirected. Self-circles are not allowed. A p-route parcel of a diagram is a mapping P: V--> [1...p] of its vertices into p subsets S1, S2…. Sp, where ÈiSi=V and SiÇSj= AE wherever i¹j. Each segment creates an arrangement of cut edges, Ec, characterized as the subset of E whose endpoints lie in particular segments Ec={(vi,vj)|(vi,vj)ÎE,P(vi) ¹P(vj)}[10,14]. The heaviness of every subset, |Si|, is characterized to be the Number of vertices mapped to that subset. Given a graph as in order, the diagram apportioning issue tries to discover a p-route package in which every subset contains generally the same number of vertices and the amount of cut edges, |EC|, is minimized. For info diagrams that speak to workload as portray in the presentation, each apportion subset speaks to calculation and information that have to be selected to an indusial processor. The cut edges speak to the bury host connection needed by the transportation[5]. In this way, the diagram parceling issue activities to discover a misuse that adjusts the calculation done by every processor while minimizing the combined between host correspondences. The established graphs partitions shares the disadvantage that stems from the previous information of the workload and system state, and cannot adapt to system changes. On the other hands, they mostly focus on tightly-coupled systems consisting of homogenous processors thus cannot handle the heterogeneity in distributed systems [7]. Additionally, these algorithms only consider the imbalance of sub domains and the edge-cut communication, while ignore the data movement cost that can be a critical performance bottleneck due to the high communication latency in distributed systems.

IV. ARTIFICIAL NEURAL NETWORKS

First we define Neurons In Artificial Neural Networks [7,8] Neurons the performance of the brain as a network of units be explain as computational model, stimulated by biological research relating to this artificial Neural Networks. Neurons are the most basic elements of the human brain which provides humans with abilities to commit to memory, think, and apply previous experiences to every action. Mostly, a biological neuron receives inputs from current neurons, each to gather them in some way, performs a commonly non-linear process going on the result, and then gives the outputs the last result [3, 6 and 15]. The advantage of the human brain comes from the large numbers of these basic connections and components between them[13]. Like brain, artificial neural networks consist of interconnected giving out units called neurons that send signals to one another depending on their incoming signals. The three basic components of the (artificial) neuron are:

1) The connecting links or synapses that provide weights, wj, to the input values,xj for j = 1, ...m;
2) An added that sums the weighted input values to compute the input to the activation function v = w0 + m_j=1wjxj, where w0 is called the bias (not to be confused with statistical partiality in prediction or estimation) is a numerical value associated with the neuron. It is expedient to think of the bias as the weight for an input x0 whose value is always equal to one, so that v = m_j=0wjxj; 3. An activation function g (also called a squashing function) that maps v to g (v) the output value of the neuron. Input vector is applied to the neuron via input connections. The links have weights which changes while the neural network learns [6, 9, and 12]. Weights can either reduce or motivate the communication of the input value. Mathematically, input values are multiplied by the value of that particular weight. At the neuron node, all weighted-inputs are summed. This summed value is then passed to an establishment function. Some examples of establishment functions are: step function, incline function and sigmoid function: Learning of load balancing is a hard task which has been investigates by several researchers.

Given a diagram as info, the chart dividing issue looks to discover a p-way the trouble is raised from the issue's intricacy. The vicinity of additional workload and equipment heterogeneity on the hosts puts additional randomness to this issue. At the point when apparatus heterogeneity is concerned, one of the inquiries that rings a bell is the thing that sizes ought to responsibility parcels be such that they coordinate the has accumulation or capacities of hosts on which they are to be mapped. Another investigation is the way these allotments mapped to has so that the normal execution time of the entire application is minimized. Prior to the responses to these and related inquiries are given, late endeavors that address burden adjusting and machine learning are talked about. Mehra and Wah [7, 9, and 13] made a device, SMALL, which uses fake neural systems as the learning device for burden adjusting. As per their work, burden adjusting issues have two parts: load records, which show every host's heap; and choice arrangements, which focus both the circumstances under which undertakings are relocated and the destinations of approaching errands. Burden adjusting frameworks use workload files to increasingly calendar assignments. Load lists are the qualities that show the amount of a host is stacked. Load files for the most part join data from the key assets of conflict: CPU, plate, system, and memory. Elective destinations for every forthcoming assignment by the normal pace ups are positioned by burden files. Ordinarily, load qualities are processed utilizing a physically indicated

equation as elements of present and late use levels of different assets. In any case, Me27 hra and Wah propose positioning option destination for an approaching errand by their individual finishing times. Be that as it may, finish times must be measured for finished errands, while choices should be made before undertakings begin. Consequently, consummation times should be predictable utilizing just the data accessible before an undertaking starts execution. Without knowing the asset necessities of assignments, supreme errand completion times can't be predictable. Consequently they propose to foresee the relative finishing times of the undertakings for diverse hosts. It is adequate that each host predicts the finishing time of an approaching undertaking with respect to its culmination time on the picked unmoving record server, given just the stacking conditions at assignment entry time. Anticipated relative fulfillment times can be utilized as burden lists. Comparator neural systems, one for every host, is utilized to figure out how to foresee load files (the relative speedup of an approaching employment) utilizing just the asset usage examples of the hosts saw before the work's landing another part of the work, Mehra and Wah discusses how decision policies can be self-learnt. In there is an example load balancing decision policy. The sender-side rules are evaluated at s, the host of arrival of a task. Reference can be either 0 or Min Load; the other parameters - d, q1, q2 - take non-negative real values. A remote destination, r, is picked randomly from Destinations, a set of hosts whose load indices fall within a small neighborhood of Reference. If Destinations is the empty set, or if the last sender-side rule fails, then the task is executed locally at s; otherwise, site r is requested to receive the task. Upon receiving that request, site r applies its receiver-side rule. If the receiver-side rule succeeds, the task is migrated In order to avoid the resource ci from getting overloaded, instantaneous job migration is used. The following describes the procedure for instantaneous job migration,

Source load = current load of the resource ci Sort L NSet in ascending order based on load If (source load > 0.97) Migrate jobs to the resource ck in SAdj Having least load Update load value of ck End if

A. Number of Jobs versus Mean Response Time

Mean Response Time: Response time rj of job j is the time period from the job arrival to the completion time of the job, i.e., the time spent in the resource queue plus the job service (execution) time. The mean response time RT,RT=1/N.

The performance of the proposed work is compared with the non relocation algorithm by varying the number of jobs. The system load is varied by varying the number of jobs submitted. The higher the load, the higher the mean response time of both algorithms. By comparing these two algorithms, it is found that there is a considerable variation in the mean response time when the number of jobs increases.

V. CONCLUSION

The existing load balancing techniques have been compared, this comparison is similar to that given in references [4][6] except that two additional matrices are added. Existing load balancing techniques that have been discussed worked in distributed, and large scale distributed network environment and mainly focus on reducing associated overhead, service response time and improving performance etc. but none of them have considered the energy consumption and carbon emission factors .Therefore, there is a need to develop an energy efficient load balancing technique that can improve the performance of cloud computing by balancing the workload across all the nodes in the distributed systems along with maximum resource utilization, in turn reducing energy consumption and carbon emission to an extent which will help to achieve Green computing. The proposed method utilizes green elements in the design and building process of a load balancing (i.e. ANN). ANN predict the demand and thus allocate resources according to demand. Thus, it always maintains the active servers according to current demand, which results in low energy consumption than the conservative approach of over provisioning. High utilization of server results in more power consumption, server running at higher utilization can process more workload with similar power usage.

REFERENCES

[1] Said Fathy El-Zoghdy. A Load balancing Policy for Heterogeneous Computational Grids Vol. 2, No. 5, 2011.
[2] S. Xian-He, W. Ming, GHS: A performance system of Grid computing, in: Proceedings of the 19th IEEE International Symposium on Parallel and Distributed Processing, 4–8 April 2003.
[3] X. Tang and S. T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. In Proc. of the Intl. Conf. on Parallel Processing, pages 373–382, August 2000.
[4] Kalamuddin Ahmad, A.A. Zilli , Mohd. Husain," A Statistical Analysis And Comparative Study of Embedded Hypercube", International Journal of Computer Applications, Volume 103, Oct 2014.
[5] Mohammad Haroon, Mohammad Husain," Analysis of a Dynamic Load Balancing in Multiprocessor System", International Journal of Computer Science engineering and Information Technology Research, Volume 3,March 2013.
[6] Mohammad Haroon, Mohammad Husain," Different Scheduling Policy For Dynamic Load Balancing in Distributed System", 3rd international conference TMU Moradabad.
[7] Mohammad Haroon, Mohammad Husain," Different Types of Systems Model For Dynamic Load Balancing", IJERT, Volume 2, Issue 3, 2013.
[8] Mohammad Haroon, Mohammad Husain," Different Policies For Dynamic Load Balancing", International Journal of Engineering Research And Technology, Volume 1, issue 10, 2012.
[9] Mohd Haroon Ashwani Singh, Mohd Arif, "Routing Misbehabiour In Mobile Ad Hoc Network", IJEMR, Volume 4, Issue 5 ,October 2014.
[10] Mohammad Haroon Dr Mohammad Husain , A A Zilli, Mohd Kalamuddin "Appearance Determined Interest Attentive Load Balancing Algorithm", International Journal of Engineering Technology, Management and Applied Sciences, Publication date 2014/12, Volume 2.

[11] Abdul Muttalib Khan, Mohd. Haroon Khan, Dr.Shish Ahmad," Security In Cloud By Diffie Hellman Protocol",International Journal Of Engineering And Innovative Technology(IJEIT), Volume 4 , Issue 5 , November 2014.
[12] Muhammed Usman Khan, Afsaruddin, Riyazuddin Khan, "Load Balancing In Distributed Network by Simulated Annealing Method", IJSRD - International Journal for Scientific Research & Development| Vol. 3, Issue 07, 2015.
[13] Mohammad Haroon, Mohd Husain,"Interest Attentive Dynamic Load Balancing in Distributed Systems",IEEE.
[14] Mohd Haroon, Mohd Husain, Manish Madhav Tripathi, Tameem Ahmad, Vandana Kumari,"Server Controlled Mobile Agent", International Journal of Computer Applications (0975-8887), Publication date 2010/12.
[15] Riyazuddin Khan, Mohd Haroon, Shahid Husain, "Adjacent Selection Method for Load Balancing in Distributed Network by Artificial Intelligence", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, (Vol. 4, Issue 8, August 2015)

