IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-4, Issue-11 E-ISSN: 2347-2693

A Hybrid Heuristic Algorithm to Enhance Load balancing in Cloud Environment

V Ravi Teja Kanakala*, K.P avan Kumar, S. Kavitha

Available online at: www.ijcseonline.org

Received: Oct/29/2016 Revised: Nov/10/2016 Accepted: Nov/24/2016 Published: Nov/31/2016

Abstract- Cloud computing, an internet based technology which provides virtualized computer resources over the internet. Distribution of the dynamic workload among the computer resources in the cloud evenly in such a way that no single node is overloaded or under loaded is called Load Balancing. An efficient load balancer will increase the performance of cloud, maximizes the cloud services and also increases the resource utilization. Today increasing the performance of a cloud depends on many factors, among them Load Balancing is one of the main factors. In this paper we propose a load balancing algorithm which is a variant to the Weight Least Connection (WLC) algorithm. The proposed algorithm shows better results in several aspects like accurate calculation of work load on a resource, distributing the work load on the service nodes efficiently, enhancing the response time and minimizing the overall task execution time.

Keywords: Cloud computing, Virtualized computer resources, Dynamic workload, Load balancer, Load balancing algorithm, Least Connection (LC) algorithm, Power consumption.

I. Introduction

A Cloud is a pool of virtualized computer resources that are dynamically scalable. The computer resources in the cloud will be allocated for executing high-scale computational tasks which is called Load balancing. A Load balancing algorithm is said to be an enhanced and effective algorithm if it shows better performance among other load balancing techniques in multiple aspects while executing a task like reducing the average execution time, enhancing the responsive time, reducing the power consumption, reducing the communication costs, executing the tasks with high priority more efficiently etc. All the above listed areas show their impact while balancing the work load in a cloud environment. There are many algorithms proposed for balancing the load in a cloud but still no algorithm is able to perform well in all the above listed aspects i.e. if an algorithm performs well in enhancing the responsive time of a cloud then the cost of communication or the power consumption will be increased in return due to increase in the usage of resources. Similarly if an algorithm executes a task with high priority first then it will affect the waiting time and responsive time of the cloud. Load Balancing is one of the main challenges that cloud computing is facing today. Efficient load balancing in a cloud will solve the maximum performance related problems. Load balancing can be done using static and dynamic methods but in cloud computing environment most cases are with dynamic workload because the data regarding the service nodes should updated frequently. So, dynamic load balancing algorithms best suits for cloud environment. In static load balancing techniques while assigning connections to the service nodes the load balancer will not consider the previous performance by the node [1]. The allotment of the connections will be purely based on the fixed information about the capacity of the node like processing power, storage capacity and memory availability etc. The problem with static load balancing is if there is a change occurs in the information collected by a node before assigning a connection then we cannot adapt the dynamic changes during runtime [3]. But in the case of dynamic load balancing algorithm the load balancer can adapt the changes frequently even at the runtime of the service node [1].

II. Related Work:

Existing Load Balancing Algorithms

Least Connection (LC) Algorithm:

In the basic Least Connection algorithm the load will be calculated based on the number of connections the resource is having with the clients which is an inefficient way of calculating the workload of a node and is far away from accuracy and not preferable [5]. It will also not check the node's capabilities like Processing power, Disk space availability, and memory etc before allocating task. The principle idea of Least Connection algorithm can be shown mathematically. Suppose there are a group of resource nodes $R= \{R_0, R_1, R_2 \ldots R_n\}$. The number of connections which the resource node R_i currently having can be represented as $C(R_i)$ where (i=0,1,2,3,…n-1). The sum of all the connections of all the Resource nodes in the cluster can be represented as $C_{sum}=\Sigma C(R_i)$ where (i=0,1,2,3….n-1). To get the newly arrived connection a Resource node R_{new} should satisfy the condition (1).

$$\mathbf{C}(\mathbf{R}_{new})/\mathbf{C}_{sum} = \min \{\mathbf{C}(\mathbf{R}_i)/\mathbf{C}_{sum}\}$$

where (i = 0,1,2,3,……n-1) (1)

Weight Least Connection (WLC) Algorithm:
Improving the Least Connection (LC) algorithm there were several algorithms proposed. One of them was Weighted Least Connection (WLC) algorithm. In Weighted Least Connection (WLC) algorithm each and every resource node will be assigned a fixed performance weight depending on its capacity like processing power, idle rate, size of memory and I/O available etc. Resource node R_{new} to get a new connection then it should satisfy the condition (2).

$$\frac{C(R_{new})/C_{sum}}{W(R_{new})} = \min \{\frac{C(R_i)/C_{sum}}{W(R_i)}\}$$

where (i = 0,1,2,3,……n-1) and $W(R_i) \neq 0$. (2)

Since C_{sum} can be considered as constant the above equation can be derived as

$$\frac{C(R_{new})}{W(R_{new})} = \min \{\frac{C(R_i)}{W(R_i)}\}$$

where (i = 0,1,2,3,……n-1) and $W(R_i) \neq 0$. (3)

The above condition means that when a new connection arrives in the ready queue for execution then it will be assigned to the resource node R_{new} with higher weights and minimum number of connections.

Problem while calculating the Weight
In Weighted Least Connection (WLC) algorithm a fixed performance weight of resource node is assigned based on its capacity which is not an efficient method because the weight of a node will not be static and constant it will vary frequently. The fixed weights of resource nodes will not reflect the actual load because every connection existing with a service node will not last for the same amount of time [6]. Some connections may be short and some may have long connectivity depending upon their load severity. Calculation of accurate weight of Resource node and scheduling the connection to its favorite resource node is very important to perfectly balance the load in the dynamic environment like cloud. If we get the accurate weights of each and every resource node it will be simple and easy to map a connection to suitable resource that can execute the tasks in a connection faster.

Existing Heuristic Algorithms for Scheduling Independent Tasks in Dynamic Environment:

Min-Min Algorithm: In this technique the minimum completion time of each and every task on the cluster of resources will be calculated and the resource on which the task has minimum completion time will be its favorite resource. Among all the tasks the task with minimum completion time will be selected first and will be allocated the resource. In this technique smaller tasks will be given high preference [7].

Max-Min Algorithm: In this technique the minimum completion time of each and every task on the cluster of resources will be calculated and the resource on which the task has minimum completion time will be its favorite resource. Among all the tasks the task with maximum completion time will be selected first and will be allocated the resource. In this technique larger tasks will be given high preference. The first few steps in Max-Min algorithm is similar to the Min-Min algorithm but while selecting the task preference will be given to the task with maximum completion time [4].

Problem of priority assignment between smaller and larger tasks:
In Min-Min algorithm the smaller tasks are being assigned their favorite resources first but while coming to the case of large tasks they were not having the choice of acquiring their favorite resources [4]. So, in Min-Min algorithm larger tasks were facing problems in resource selection which will affect the total make span of larger tasks [8]. Now, considering the Max-Min algorithm larger tasks are being assigned their favorite resources which was a problem to the smaller tasks for waiting in the queue until the larger tasks are executed.

III. Proposed Algorithm:

We propose a hybrid heuristic algorithm that overcomes the above addressed problems in WLC and Min-Min, Max-Min algorithms. Assume a cluster of cloud resources $R=\{R_0,R_1,R_2,\ldots\ldots R_n\}$ and a set of connections $C=\{C_0,C_1,C_2\ldots..C_n\}$ waiting in the ready queue for execution**,** each resource node will have some capacity to execute the tasks. Let us consider the current capacity of executing tasks as weight of the resource node. Calculation of the accurate weight of resource nodes was the issue in the above described algorithms. In Least Connection algorithm (LC), the number of connections is considered as weight of the resource node which is not correct. In Weighted Least Connection (WLC) algorithm, fixed performance weights are considered which does not suit for dynamic environments like cloud. In this proposed algorithm we considered a set of multiple factors for calculating the weight of resource node such as Total number of Existing Connections (TOC), current idle rate of the node (IR), execution cost per instruction (δ), Current Complexity Rate (CR). All this data will be updated in a queue frequently after assigning every connection to nodes. The weight of the resource node can be calculated as the below

$$W(R_i) = f(TOC,IR,\lambda,CR)$$

where (i=0,1,2,3,….n-1)

$$W(R_i) = P_1 * TOC(R_i) + P_2 * IR(R_i) + P_3 * \delta(R_i) + P_4 * CR(R_i) \quad (4)$$

Where P_i is the varying parameter for the applied factors of the resource node and R_i represents the current Resource node where (i=0,1,2,3,….n-1). Along with the weight of Resource nodes we also need to calculate the weight of the connections in ready queue for assigning them to the resources that are more suitable and can handle their complexity more efficiently. While calculating the Weight of a connection we consider factors like Total Number of Instructions in the connection (TOI), Complexity of Instructions (COI), Arrival time of connection in the ready queue (AT) and also the Living Time of the connection in the ready queue (LT) which means if the connection is not assigned to any of the resource node within the Living time it will be moved into dead state. The Weight of the connection can be represented by the function (5).

$$L(C_i) = f(TOI,COI,AT,LT)$$

where (i=0,1,2,3,….n-1)

$$W(C_i) = P_k * (TOI(C_i) + COI(C_i) + AT(C_i) + LT(C_i)) \quad (5)$$

Where P_k is the constant parameter for the applied factors of the resource node and R_i represents the current Resource node where (i=0,1,2,3,….n-1). The above calculated set of data about the Weight of Resource nodes R_i and Weight of connections C_i will be implemented in queues. After the assignment of each connection to node the queues will be updated each time. By updating the queues the weights of Resource nodes will be more accurate and will help in balancing the load more efficiently. Getting accurate load value alone is not sufficient for efficient load balancing. We also need not map the tasks to the resources which can handle the task complexity efficiently and which can reduce the execution time, response time etc. So, we need to schedule data in such a way that will not create load imbalance in the system and should not require frequent migration of tasks from one resource to other for balancing the load.

Calculate the average weight of all the unassigned connections $W(C_{avg})$ in the ready queue and make it a threshold point. Consider the connections with weights greater than the threshold value as larger tasks and the weights less than the threshold value as smaller tasks. If the connection waiting in the ready queue is smaller task i.e. having lesser weight than the average threshold value $W(C_{avg})$ then apply Min-Min algorithm, then the connection with minimum execution time will be given priority and it will be allocated its favorite resource node. If the connection waiting in the ready queue is larger task i.e. having more weight than the average threshold value $W(C_{avg})$ then apply Max-Min algorithm, then the connection with maximum execution time will be given priority and it will be allocated its favorite resource node. Each time when you assign a new connection to a resource node we need to update all the queues maintaining the details about the weights of connections and resources. Before assigning the next connection the average weight of all the unassigned connections $W(C_{avg})$ must be calculated again because after the assignment of a connection in the previous step it will be deleted from the ready queue and there will a change in the average weight of connections. So, we need to calculate $W(C_{avg})$ of the remaining connections in the ready queue after every assignment.

$$W(C_{avg}) = \frac{\sum W(C_i)}{Total\ number\ of\ unassigned\ connections}$$

where (i=0,1,2,3,….n-1) (6)

IV. **The Proposed algorithm is as shown below:**

Step 1: Start

Step 2: Consider a cluster of resource nodes R_i and the set of connections C_i.

Step 3: Calculate the weight value of resource nodes as describe above $W(R_i)$ and store it in a Queue.

$$W(R_i) = P_1 * TOC(R_i) + P_2 * IR(R_i) + P_3 * \delta(R_i) + P_4 * CR(R_i)$$

Step 4: Calculate the accurate Load value of connections $L(C_i)$ and store it in a Queue.

$$W(C_i) = P_k * (TOI(C_i) + COI(C_i) + AT(C_i) + LT(C_i))$$

Step 5: Calculate the average weight of all the unassigned connections in the Ready queue

$$W(C_{avg}) = \frac{\sum W(C_i)}{Total\ number\ of\ unassigned\ connections}$$

Step 6: Now compare the next connection in the ready queue with the average weight of All the Connections $W(C_{avg})$.

if $W(C_i) < W(C_{avg})$

{

apply Min-Min algorithm;

Select the task with minimum execution time required;

Allot its favorite resource;


```
}
        if W(Ci) > W(Cavg)
        {
         apply Max-Min algorithm;
         Select the task with maximum execution time required;
    Allot its favorite resource;
    }
```

Step 7: Update the queues maintaining the weights of resources and connections after every assignment of a connection.

Step 8: Calculate the average weight of all the unassigned connections $W(C_{avg})$ after every assignment of a connection (step 5).

Step 9: If the Queue of connections is not empty repeat **step 3** else **step 10**

Step 10: End

V. Future Work:

In future we will implement the above proposed algorithm using the simulation tool Cloud Sim and provide the results. In the above proposed algorithm lot of communication happens between the connections in the ready queue, Resource nodes and the load balancer. This will increase the total communication cost which will be a drawback for the system. In future we will try to reduce the communication cost without degrading the performance

VI. Conclusion:

The proper assumption of the weight of a resource node in a dynamic environment like cloud computing is little bit complex. In this algorithm multiple factors of Resource node are taken into account for getting accurate value for the weight of Resource node. Similarly, before allotting connection complete requirements of that connection are analyzed. The goal of this algorithm is to distribute the workload efficiently among the resource nodes which can be achieved by the proper calculation of the weights of Resource node. With proper assignment of connections to the resource nodes the overall task execution time will be minimized and the response time will also be enhanced. In the above algorithm we proposed a technique to satisfy all the above mentioned requirements.

VII. References

[1]. Xiaona Ren, Rongheng Lin,Hua Zou, "A Dynamic Load Balancing Strategy For Cloud Computing Platform Based On Exponential Smoothing Forecast", IEEE ccis2011, pp-220-225, 2011.

[2]. Li-Hui Yang, Sheng-Sheng YuA, "Variable Weighted Least-Connection Algorithm For Multimedia Transmission", Journal of Shanghai University, Volume 7, Issue 3, pp 256-260, 2003.

[3]. Shanti swaroop moharana, rajadeepan d. Ramesh, digamber powar, "Analysis Of Load Balancers In Cloud Computing", International Journal of Computer Science and Engineering (IJCSE) ISSN 2278-9960 Vol. 2, Issue 2, May 2013.

[4]. E. Kartal Tabak, B. Barla Cambazoglu, Cevdet Aykanat, "Improving the Performance of Independent Task Assignment Heuristics MinMin, MaxMin and Sufferage", IEEE Transactions On Parallel And Distributed Systems, VOL. 25, NO. 5, MAY 2014.

[5]. Ra´ul Alonso-Calvo, Jose Crespo, "On distributing load in cloud computing: A real application for very-large image datasets" Procedia Computer Science 1, Elsevier, 2669–2677, 2012.

[6]. Kousik Dasgupta, Brototi Mandal, Paramartha Dutta, Jyotsna Kumar Mondal, Santanu Dam, "A Genetic Algorithm (GA) based Load Balancing Strategy for Cloud Computing", Procedia Technology 10, Elsevier, 340 – 347, 2013.

[7]. O.H. Ibarra and C.E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," J. ACM, vol. 24, no. 2, pp. 280-289, 1977.

[8]. S. Parsa and R. Entezari-Maleki, "RASA - A New Grid Task Scheduling Algorithm," Int'l J. Digital Content Technology and Its Applications, vol. 3, no. 4, pp. 91-99, 2009.

Authors Profile

Mr.V Raviteja Kanakala pursed B.Tech from JNTUK University, Kakinada in 2011 and M.Tech from VIT University, Vellore in year 2013. He is currently working as Assistant Professor in Department of Computer Science & Engineering, Pragati Engineering College, Surampalem since 2016. He is a life member of CSI since 2015. He has published more than 5 research papers in reputed international journals indexed in Sopus and conferences including IEEE and it's also available online. His main research work focuses on Load balancing in Cloud Computing and IOT. He has 3 years of teaching experience.

Mr.K Pavan Kumar pursed B.Tech from JNTUK University, Kakinada in 2009 and M.Tech from JNTUK University, Kakinada in year 2016. He is currently working as Assistant Professor in Department of Computer Science & Engineering, Pragati Engineering College, Surampalem since 2016. His main research work focuses on Cloud Computing.

*Mrs.S Kavith*a pursed B.Tech from JNTUH University, Hyderabad in 2003 and M.Tech from JNTUK University, Kakinada in year 2009. She is currently working as Assistant Professor in Department of Computer Science & Engineering, Pragati Engineering College, Surampalem since 2016. Her main research work focuses on Cloud Computing. She has 13 years of teaching experience.

