International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-2, December 2019

An efficient Task Scheduling and Load Balancing in Cloud Computing using KD-Tree Algorithm

Usha Kirana S P, Demian Antony D'Mello

Abstract*: Cloud Computing provides the sharing ability and access for available cloud host and various distributed environments, namely Load Balancing (LB), virtualization technologies and scheduling techniques. The satisfaction of both users and cloud providers are the major issues for effective LB and task scheduling algorithms in cloud resource management, where the requirements namely high resource utilization, low monetary costs and minimum makespan. Many researchers tried to develop various heuristic and meta-heuristic algorithms to attain the aforementioned user requirements. But, when the number of tasks grows exponentially, these algorithms failed to achieve LB, lower running time, and it faces the high time complexity. In this research work, a KD-Tree algorithm is developed to address the issues of heuristic algorithms and provide efficient LB by partitioning the environments into several tasks. According to the deadline of task execution, the remaining tasks are adjusted dynamically by the proposed KD-tree algorithm in the virtual environment. The experiments are conducted to evaluate the efficiency of KD-Tree algorithm with existing heuristic techniques by using makespan, energy consumption and task migrations. When the number of tasks is 20, the proposed KD-Tree algorithm achieved 71.33% makespan and 5% task migrations.

Index Terms*: Cloud Computing, Heuristic methods, KD-Tree algorithm, Load Balancing, Makespan, Time Complexity.

I. INTRODUCTION

Cloud Computing (CC) allows end users to access the applications and associated data from anytime and anywhere. In addition, the on-demand resources are obtained by an end user from the CC [1,2]. This paradigm provides the computation, software services and storage applications, which can be accessed by the end users on the basis of on-demand. Custom applications with high scalability and availability are provided and offered by various CC providers namely Google Compute Engine (GCE) and Amazon Web Services (AWS) [3,4].

Revised Manuscript Received on December 30, 2019.
***** Correspondence Author
Usha Kirana S P*, Department of Computer Science and Engineering, Visvesvaraya Technological University, Belagavi, India. Email: usha14.nayak@gmail.com
Demian Antony D'Mello, Department of Computer Science and Engineering, Canara College of Engineering, Mangaluru, India.

The computing resources are presented to clients by running the physical machines into multiple Virtual Machines (VMs) in a cloud environment. But, cloud providers are aimed to increase their profits as much as possible by misbehaving to the end users. Moreover, the providers' workloads and amount of resources are changed over time. Therefore, in such cases ensuring the profit of providers and the satisfaction of user's Quality of Service (QoS) requirements are the main issues, which should be considered [5,6]. To satisfy the customers of CC, QoS plays an important role by providing better services to the end users [7]. The number of Service Level Agreement (SLA) violations are reduced and provides better QoS services to the end customers by using efficient scheduling and LB algorithms. The execution time of requested applications of users are speeded up by using an efficient mechanism of LB algorithms, which also give fair access to end users and reduces the system imbalance in an effective way [8,9].

There are two categories of LB algorithms presents in CC such as static and dynamic LB [10], where various meta-heuristic algorithms namely Artificial Bee Colony (ABC), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Genetic Algorithm (GA) [11-13] are developed to achieve better LB. Although, meta-heuristic algorithms attained LB effectively, it renders less useful to end users in realistic computational infrastructures due to high time complexity. The task's execution time is reduced and resource utilization are maximized by using various types of priority methods namely VM priority, task priority and sorting methods, that are involved in several heuristic algorithms [14,15]. In this research work, KD-Tree Algorithm is developed and implemented to balance the workloads among resources by partitioning the virtual environment. The LB is trigged when the server is overloaded and then the region limit is readjusted by changing the split coordinates, that are stored in the KD-Tree. In simulation testing, the prototypes are developed and used for testing their efficiency, where the performance of KD-Tree is compared to existing heuristic algorithms in terms of makespan, task migration and energy consumption. The load of a VM is identified by using the threshold value of KD-Tree algorithm. The task is removed in the particular VM which is having overloaded, and according to the deadline of task execution, a task is allocated to the identified VM.

The rest of the paper is organized as follows, Section II presents the detailed research of existing LB techniques used in CC. The contribution of this work and the proposed methodology are described to solve the issues of scheduling and LB is explained in Section III and IV. The performance of KD-Tree is compared to existing techniques by conducting several experiments are discussed in Section V. Finally, the conclusion of the paper with future work is detailed in Section VI.

II. LITERATURE WORKS

In CC, LB is considered as a NP-hard problem. Several researchers are tried to address these problems in CC. Therefore, the best optimum solution is needed to compute the amount of computation time. In this section, the research of recent developments in LB algorithms are discussed with its advantages and limitations K. R. Babu, and P. Samuel, [16] solved the problem of LB by developing the Enhanced Bee Colony (EBCA) Algorithm in cloud environment. This method also considered the priorities of tasks in the waiting queues of VM. The imbalance was reduced by selecting the tasks with minimum priority for migration. To validate the effectiveness of these methods, the experiments results used the makespan and a number of migrations as parameters against traditional BCA. The performance of the entire cloud eco system is degraded due to frequent migration of tasks.

M. Lawanyashri, et al., [17] implemented the hybrid optimization algorithms includes Fruitfly with Simulated Annealing Algorithm (FOA-SA) for improving the optimum usage of VM, convergence rate, execution time and optimization accuracy. The makespan, energy consumption and cost of data center were reduced by balancing the workload with dynamic threshold values in FOA-SA. The energy consumption was reduced by using the sleeping strategy of FOA-SA, but the method failed to concentrate on network traffic information, which was an important issue in CC. T. Jena, and J. R. Mohanty, [18] implemented the Genetic Algorithm-based Customer-Conscious Resource Allocation and Task Scheduling (GACCRATS) to fulfil the gap between available infrastructure and frequently changing requirement of customer. The makespan time was reduced and achieved the maximum number of customer satisfaction by mapping the tasks to VM of multi-cloud federation. The experimental results showed that the GACCRATS method increased the scalability of multi-cloud environment. Even though, the method achieved good performance, the energy consumption, latency arbitration, running cost and data locality cost were not considered by GACCRATS.

P. Durgadevi, and S. Srinivasan, [19] developed the combination of optimization techniques namely Cuckoo Search and Shuffled Frog Leaping Algorithm as SFLA-CS algorithm for resource allocation. These algorithms were used in complex situations due to an easier evaluation, where the sizes and request speed were analyzed. The resources were allocated on the server side by using these easier evaluations and the computed time was consumed very less. The high speed convergence was achieved by using the capacity of global optimization. The execution time, throughput, turnaround time and percentage of allocation were high in SFLA-CS method when compared with other existing techniques such as ABC and CS. L. Kong, et al., [20] developed a fast heuristic algorithm based on the zero imbalance method to address the issues of NP-problem in a heterogeneous environment. The completion time difference was minimized by using these method without taking complex scheduling decisions and priority methods. The LB and task scheduling were achieved effectively by defining two constraints, namely the earliest finish time and optimal completion time. The various performance metrics such as degree of imbalance, resource utilization, makespan, waiting time, monetary cost, running time (scheduling time) and efficiency were used to analyze the performance of this algorithm. But, the method failed to focus on power consumption, which was the major impact on the efficiency of LB. J. P. B. Mapetu, et al., [21] scheduled and balanced the tasks by developing the binary version of PSO as BPSO with low cost and low time complexity. The execution cost was reduced and avoided the under-loaded VM as well as over-loaded VM by improving the updating method for particle position. By using BPSO, the time complexity and waiting time of user request was reduced and also maintained the system scalability by scheduling the tasks effectively in cloud resource management. The experimental results validated the effectiveness of BPSO by using several parameters namely Degree of imbalance, makespan, resource utilization, execution cost, running time and waiting time against several heuristic algorithms. In order to satisfy both cloud users and providers, the performance of the cloud and LB are influenced by live migration and energy consumption.

III. CONTRIBUTION OF THIS RESEARCH WORK

When compared with the existing works, the major focus of this research work is to find the requirements of providers and users with lower computational time. The time complexity of scheduling algorithms, load imbalances, wait times are minimized and the resource utilization are maximized by cloud providers for gaining much profit as well as increasing the number of user tasks. The low monetary cost, less makespan time and highest availability are required by the cloud users. This paper develops the efficient and fast KD-tree algorithm by dividing the tasks to balance the workloads in contrast with several heuristic algorithms. The major specification of the work is discussed as:

- The completion time difference is minimized among the heterogeneous VMs running and mapping in parallel on various host, which is used to balance the requirements of providers and users by designing the KD-Tree algorithm, instead of using complex heuristic algorithms.
- The extensive experiments are carried out using real workloads to validate the feasibility of proposed KD-Tree algorithm, when compared with existing scheduling algorithms.

IV. PROPOSED METHODOLOGY

The virtualized data centers support the CC, which is an attractive platform for various diverse applications because it provides the services to end users. In a cloud platform, many researchers tried to develop an efficient task scheduling and LB mechanisms, but they are unable to consider the service providers' perspective as well as optimum metrics from the users.

This research work is developed to provide the benefits to both users and providers by assigning the jobs towards dynamic provisioning of resources, which leads to better makespan and migration time. The tasks are considered as node in this proposed work. The cloudlets will be assigned in VMs, when the nodes are divided for task execution. Sometimes, the VMs are classified into under loaded and overloaded due to the variations in processing capacity for several VMs. In such cases, the better performance is achieved by developing the efficient LB algorithms. The tasks are migrated because some VMs are overloaded with multiple tasks, where these migrated tasks are assigned to under loaded VMs. According to the priority, the migrated tasks are selected in this case. The lowest priority tasks are used as a candidate for migration process, in which Fig. 1 shows the basic structure of LB method.

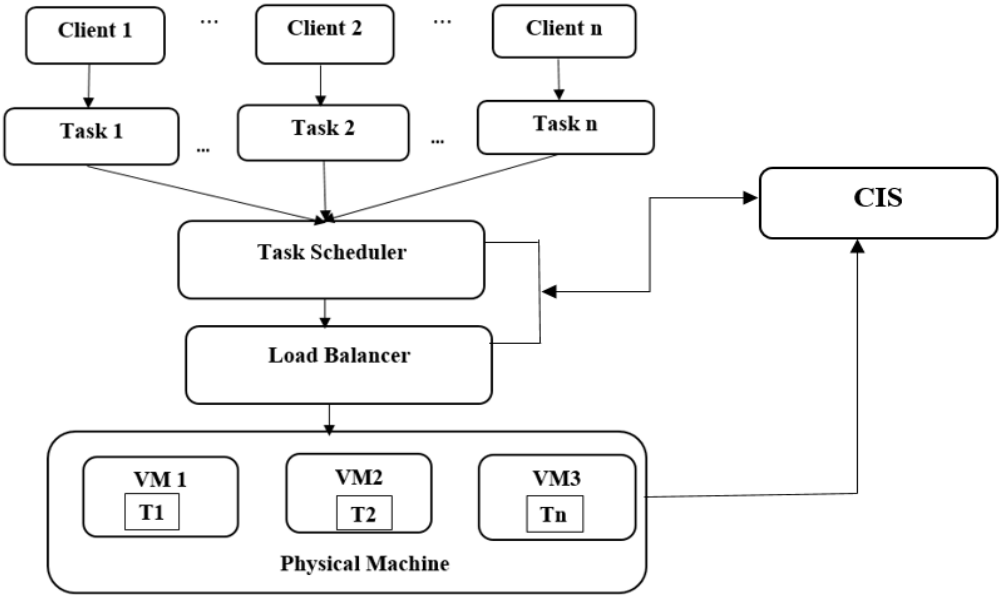

Fig. 1. Basic Structure of LB method

In the cloud environment, all the resources are available in the repository called Cloud Information Service (CIS), where it is also defined as registry of data centers and it should register to CIS, when it is newly created. Normally, data centers are heterogeneous in nature due to the specific characteristics, which consists of various hosts. The number of processing elements with RAM and bandwidth characteristics are presented in the hosts. According to user request, different number of VMs are virtualized from those hosts. As like hosts, the VMs also have heterogeneous characteristics. In the data centers, the information about all the resources are collected by CIS. According to these information, the tasks are submitted to various VMs by cloud broker, where this research work considers the load balancer as well as task scheduler as a cloud broker. The overloaded conditions are verified by using the proposed algorithm and the tasks are migrated to under loaded VMs from overloaded VMs.

Consider a cloud computing environment with heterogeneous users with varied job requirements. The provisioning of the resources results in the following overhead.

Let $U = \{u_1, u_2, u_3, ..., u_M\}$ be the number of users.

$C = \{dc_1, dc_2, dc_3, ..., dc_N\}$ cloud consists of group of datacenters $dc_1 \, to \, dc_N$

$H = \{h_1, h_2, h_3, ..., h_L\}$ be the number of hosts (servers) in each data center. $V = \{v_1, v_2, v_3, ..., v_k\}$ be the number of VM in each host. When a user submits a job, the following activities are carried out, which is also shown in Fig. 2:

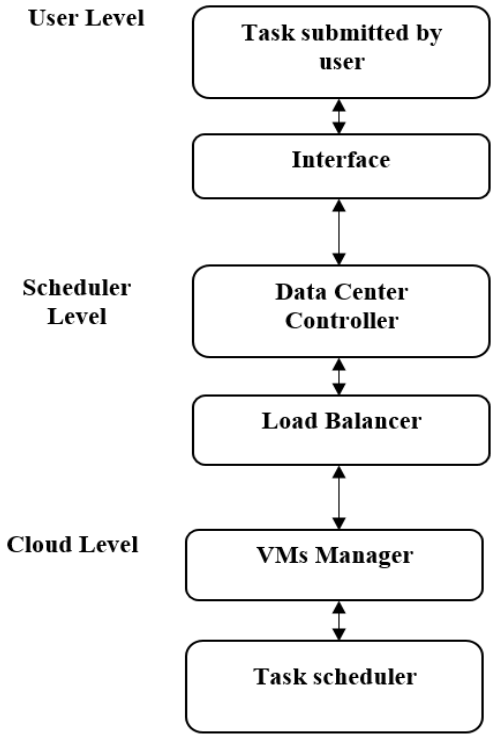

Fig. 2. Diagrammatic representation of Problem Definition

- **User-Level:** Cloud users submits tasks
- $T_1, T_2, ..., T_Z$ to task handler which is supported by data center controller.
- **Scheduler Level:** This level implements a cloud data center controller that accepts user submitted requests and maps to the task scheduler. The jobs are queued for a regular interval of time are filtered by the job size predictor which will arrange the tasks in the sorted order as per their Million Instruction Per Second (MIPS) requirement and the response time constraint and forms the task groups for allocation to the individual clusters. The clusters may be homogeneous or heterogeneous with different capabilities like low, medium and high capabilities. Load Balancer is used to minimize the waiting time and completion time of the tasks. However, improper task assignment strategy may unbalance the load among the VMs, i.e., few VMs are heavily loaded and others are idle and it also increases the completion and waiting time of the tasks. As a result, the overall make span of the system also increases. Hence, the proper LB among the VMs becomes a critical task to improve the performance of the system resources and maintain the stability of the environment.
- **Cloud-level:** VM manager assigns a task to available virtual machine using to balance the load**.**

A. Proposed KD-Tree Algorithm

The point data are distributed in a k-dimensional space, where these data are indexed by a data structure called KD-Tree, which is also considered as k-dimensional binary search tree. To solve the space partitioning problems, algorithmic solutions are developed i.e. KD-trees. The two smaller sub-sections are derived along with x-axis and y-axis from the regions, which is used to create the KD-trees. In this section, the proposed LB algorithm called KD-Tree is described. The proposal is based on two criteria: first, the load of a server is presented as the bandwidth to send state updates to clients; and second, the system should be considered heterogeneous (i.e., every server may have a different amount of available bandwidth). Each node of the tree represents a region of the space defined by a coordinate used to split the space. The two half-spaces are represented by every two children of that node, where the half-space is obtained in the partitioning process.

The splitting axis is alternated in the case of two dimensions, the axes x and y at each level of the tree, where if the first-level nodes store x-coordinates, the second-level nodes store y-coordinates. Every leaf node represents a region of the space, but it does not store any split coordinates. Instead, it stores a list of the tasks present in that region. Finally, each leaf node is associated to a server. When a server is overloaded, it triggers the load-balancing, which uses the KD-tree to readjust the split coordinates that define its region, thus reducing the amount of content. The nodes of the KD-tree also store two other values: capacity and the load of the sub-tree. The load of a leaf node is equal to the load of the associated region. Similarly, the capacity of a leaf node corresponds to the network capacity of its associated server. For each non-leaf node, the load and capacity values correspond to the sum of those stored in its child nodes. Hence the tree root stores the total weight and the total capacity, then upload bandwidth of the server system. In the following sub-sections, the construction of tree, load calculations are associated with each server and the effective LB algorithm are described.

- **KD-Tree Construction**

A balanced tree is constructed by initializing the construction of KD-tree. The tasks id values are used to verify whether each node has children or not. Each leaf represents the server associated region, which is determined in the leaf node. The difference between the number of leaf nodes in the two sub-trees of any given node is at most one, which allows to create the balanced tree.

- **Load Calculation**

Along with a specific dimension, each level of KD-Tree splits the all tasks using hyperplane, which is perpendicular to the corresponding axis. According to threshold values, all tasks will be split at the root of the tree, i.e. the tasks will be in the left sub-tree, when the threshold value is less than root and if it greater than root, the tasks will be placed in the right sub-tree. This process continues, until all the tasks have been executed. By using this partition steps, a KD-Tree algorithm effectively calculates the load of the tasks and assigned these tasks to corresponding server for execution.

- **Dynamic Load Balancing**

The various servers (i.e. under loaded servers) are assigned with particular tasks, where these tasks are migrated from the overloading servers.

To achieve this process, the data are collected and migrate to the other servers by overloaded servers using KD-tree algorithm that adjust the split coordinates of the regions. An array with tasks are maintained by each server, which is located in its respective region and sorted by x coordinates. A pointer with linked lists are stored by each element of the array, which is sorted by y coordinated tasks. The time required for balancing the load is minimized by using a local sorted tasks list on every server, since there is no need for the server to again sort the tasks lists send by other servers.

- **Task Scheduling**

Before initiating LB, the system has to find the demand to each overloaded VMs and supply to the under loaded VMs. Here the VMs are sorted based on the capacity in ascending order. The task migration is performed only when demand meets the supply. From the under loaded VM set, the proposed method selects a VM which has highest capacity as target VM. The method selects the task with lowest priority from an overloaded VM and it is rescheduled to an under loaded VM with maximum capacity.

Supply to a particular VM is the difference between its capacity and current load and it can be calculated using Eq. (1)

$$Supply_{VM} = Capacity - Load \quad (1)$$

Where, the capacity and load can be identified by using the following Eq. (2) and (3). The total capacity of the entire datacenter can be calculated from using the Eq. (2).

$$Capacity_C = \sum_{vm=1}^{n} Capacity_{VM} \quad (2)$$

As like capacity, the total load on a datacenter is the sum of load on each VMs. The equation is given by the Eq. (3).

$$Load_C = \sum_{vm=1}^{n} Load_{VM} \quad (3)$$

Then, the demand of a VM is calculated using the Eq. (4)

$$Demand_{VM} = Load - Capacity \quad (4)$$

The proposed KD-Tree algorithm is presented in the above section. On submission of each task into the cloud, the VM will measure the current load status and calculates SD. If the variation of loads is greater than the threshold, then LB process is initiated. During this LB process, VMs are classified into under loaded and overloaded VM sets. Then the submitted tasks are rescheduled to the VM having highest capacity.

V. RESULTS AND DISCUSSION

In this section, the validation of proposed KD-Tree algorithm is verified by conducting several experiments on simulated cloud environment against existing techniques such as EBCA [16] and FOA-SA [17]. The first part discussed the experimental setup of KD-Tree algorithm, where the analysis is presented in the second part.

A. Experimental Setup

The KD-Tree LB method is experimented in a personal computer with Intel Core i3 processor and 2GB memory using Windows 8 operating system. The KD-Tree method is implemented using Java with cloudsim and the performance is evaluated with various cloud set up for load and capacity. In this heterogeneous environment, number of tasks and cloudlets with different specifications are considered. In the cloud environment, VMs with varying specifications are submitted. While compared with existing methods, a number of tasks migrations and energy consumption are measured. Table 1 explains the parameters and its instances, which are used for implementation.

Table I. Parameters and its instances

Parameter	Value of Parameter
Number of cloudlets	10,15,20,25,30
Number of tasks	10,15,20,25,30
Number of VMs	3
Structure of the datasets	(Number of tasks) * (Number of VMs)
Reserved instances	i1, i2, i3, i4

B. Performance Metrics

In this section, the performance of KD-Tree is validated by using three metrics such as makespan, energy consumption and tasks migration, which are explained as follows.

- **Makespan**

The overall completion time of tasks as T_z in VM as V_k are defined as makespan. The main aim is to reduce the makespan of tasks. The mathematical expression for makespan is given in the Eq. (5)

$$F_1(Y) = Minimize\left\{ \max_{t_i \in T_Z, V_k \in V} ft_{i,k} \right\} \quad (5)$$

Where, $ft_{i,k}$ is the finishing time of task t_i on the VM V_k

- **Energy Consumption**

The energy consumption rate of VM, execution time of tasks and the energy consumption produced by the task running on the VM are used to identify the energy consumption of the task. The Eq. (6) explains the energy consumption of the tasks.

The energy consumption is identified by

$$econ_i = econ - rate_j \times exec(T_z) \quad (6)$$

Where, $econ_i$ represents the task running on VM, which produced the energy consumption, $econ - rate_j$ describes the VMs' energy consumption and $exec(T_z)$ presents the Tasks' execution time.

The total energy consumption is evaluated by Eq. (7)

$$E(X) = \sum_{i=1}^{Z} \sum_{j=1}^{n} econ_{ij} \quad (7)$$

The objective function is to minimize the energy consumption of the tasks, which is explained in Eq. (8),

$$F_2(Y) = Minimize\{E(X)\} \quad (8)$$

C. Performance of KD-Tree Algorithm in terms of Makespan

In this section, the performance of proposed KD-Tree is validated against EBCA and traditional BCA [16] in terms of makespan are discussed. Table 2 presents the experimental values of KD-Tree for makespan, where Figure 3 shows the graphical representation for makespan. The best values are represented in bold.

Table II. Computation of Makespan for Proposed against existing techniques

No. Of Cloudlets	Bee Colony (s) [16]	Enhanced Bee Colony (s) [16]	KD-Tree-LB (s)
10	50.1	43.85	40.66
15	70.1	68.85	52.66
20	80.1	78.85	71.33
25	110.1	100.1	91.33
30	120.1	118.85	102

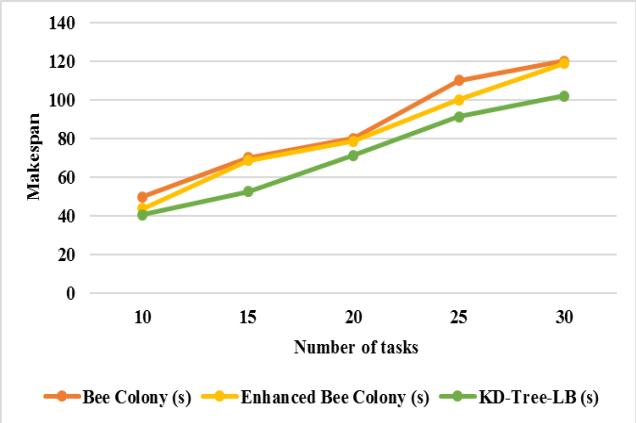

Fig. 3. Performance of KD-Tree in terms of Makespan

The Fig. 3 clearly stated that the proposed KD-Tree Algorithm provides better performance for minimizing the makespan, when compared with traditional BCA and EBCA. When the number of tasks is 15, the makespan for BCA (i.e. 70.1) is very high when compared with EBCA (i.e. 68.85). But, the proposed KD-Tree achieved very low makespan (i.e. 52.66) for the same number of tasks. When the number of tasks increases, the makespan for all algorithms are also increases gradually. For instance, the proposed KD-Tree achieved 102 makespan for the 30th tasks, however the EBCA and BCA achieved 118.58 and 120.1 makespan for the same 30th tasks. While using the proposed KD-Tree algorithm, the makespan is reduced into a significant amount. i.e. The faster response is obtained by the users when compared with traditional techniques. The service providers provided the response time faster, which is a good measure of QoS. Therefore, it is proved that the customers obtained good QoS from the providers.

D. Performance of KD-Tree Algorithm in terms of Task Migrations

The performance of the clouds is affected, when the number of tasks migrations are greater, which will automatically reduce the QoS. Therefore, the number of tasks migrations are reduced by developing the efficient LB and scheduling mechanism. This proposed KD-Tree is implemented for analyzing the number of task migrations, where the experimental results are tabulated in Table 3. The best results are provided with bolded values.

Table III. Task Migration Values for Proposed KD-Tree against existing Bee Colony

No. Of Cloudlets	Bee Colony [16]	Enhanced Bee Colony [16]	KD-Tree-LB
10	2	2	2
15	4	3	3
20	7	7	5
25	12	11	7

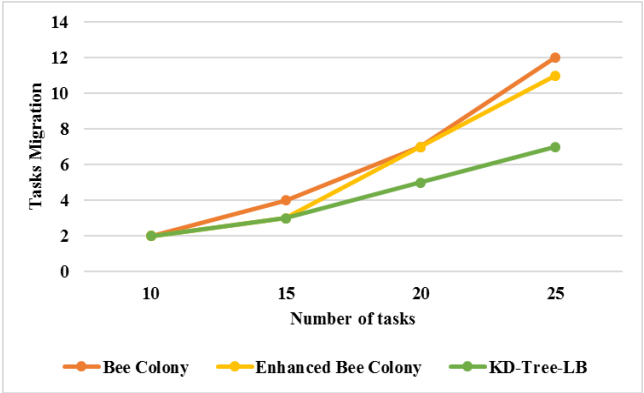

Fig. 4. Tasks Migration of Proposed KD-Tree

Fig. 4 shows the graphical representation for task migration for various number of cloudlets (tasks). When the number of tasks are less, all the algorithms are performed in the same way. For instance, the KD-Tree, EBCA and BCA provide the migration of tasks as 2 for the 10th task. But, when the number of tasks increased, the migration for KD-Tree algorithm provided better performance than all other techniques. i.e. KD-Tree achieved 5 migrations for 20th tasks, but the EBCA as well as BCA achieved 7 migrations for the same tasks. From the Table 3 and Figure 4, it is clearly proved that the number of task migrations is reduced by using KD-Tree algorithm. But, the performance of cloud eco system are affected, when the frequent migration of tasks occurs. In the next subsection, the energy consumption of KD-Tree algorithm is explained with validated results and their resultant graphs are discussed.

E. Performance of KD-Tree Algorithm in terms of Energy Consumption

In this section, the energy consumption for various tasks is calculated and the validated results are presented in Table 4. The energy consumption of proposed KD-Tree algorithm is compared with existing techniques namely PSO, Honeybee (HBB), Energy-aware FOA as EFOA and FOA-SA [17]. The graphical representation for energy consumption is represented in Fig. 5.

Table IV. Table 4: Energy Consumption for KD-Tree algorithm against existing techniques

No. Of Cloudlets	PSO (kWh) [17]	HBB-LB (kWh) [17]	EFOA-LB (kWh) [17]	FOA-SA-LB (kWh) [17]	KD-Tree-LB (kWh)
100	1.78	2.44	1.17	1.13	0.82
200	3.12	3.66	1.28	1.20	1.09
300	4.34	4.44	2.21	2.01	1.56
400	7.21	5.5	3.20	3.08	2.47

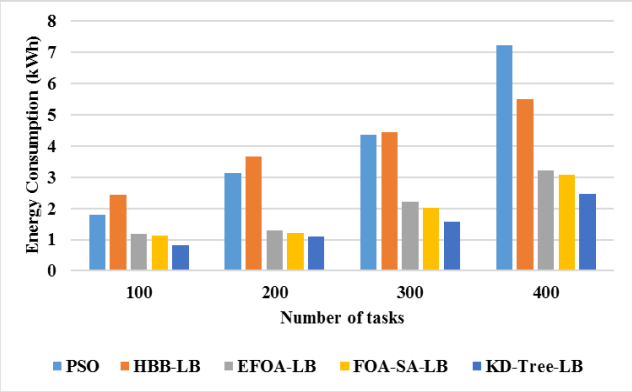

Fig. 5. Total Energy Consumption of KD-Tree Algorithm

Here, the energy is calculated in kWh. The KD-tree consumed 2.47 energy, EFOA and FOA-SA consumed nearly 3.25 energy for the 400th tasks, but PSO consumed more energy (i.e.7.21) and HBB consumed 5.5 energy for the same number of tasks. When the number of tasks is less, the energy consumption is also less for all the algorithms. For instance, all existing algorithms except HBB consumed nearly 1.80 energy, but the proposed KD-Tree consumed only 0.82 for the 100th tasks. In this proposed method, according to the threshold values of KD-Tree, the workloads are assigned in a balance state to every VMs in the data center. The tasks are removed from the overloaded VM and assigned to a particular VM, when the load is greater than the lower threshold value. Thus, it will make the KD-Tree algorithm consumes less energy when compared with all other existing techniques.

From the all experimental results, the proposed KD-Tree algorithm reduces the makespan, energy consumption and number of task migrations, when compared with existing algorithms namely PSO, BCA, EBCA, FOA-SA, HBB and EFOA. In the cloud environment, the computational resources are effectively and efficiently used due to the performance of KD-Tree algorithm. Therefore, the end users experience better QoS by minimizing the number of task migrations, energy consumption and makespan.

VI. CONCLUSION

The number of tasks are increasing exponentially, due to a vast amount of cloud users in CC. The under loaded and heavy loaded conditions in the data centers are assigned to servers by using efficient LB algorithms. In order to maintain the QoS, the tasks are migrated to under loaded VMs from the overloaded VMs of the same datacenter. These tasks are scheduled and balanced among various heterogeneous environments. But, VMs faces several constraints namely high resource utilization, low makespan, low scheduling time and execution cost. This research work introduces the KD-Tree algorithm due to the inefficiency of heuristic algorithms. The virtual environment is divided and adjusted the split coordinates to provide the LB of servers, which are stored in nodes. The fine granularity of the load distribution is allowed by making the use of KD-Tree algorithm and readjust the regions by recursively traversing the tree, when compared with other common methods. The loads are assigned to every server that is close to the ideal value, therefore, the number of migrations are reduced using finer granularity. The experimental results showed that the proposed KD-Tree algorithm achieved 2% task migration and 40.66% makespan for the 10th tasks, where it consumed only 0.82 energy for 100th tasks. In this research work, the KD-Tree algorithm effectively balances the loads and schedules the tasks for a particular VM. In future work, the KD-Tree algorithm tried to reduce SLA violations by using effective scheduling algorithms.

REFERENCES

1. P. Pradhan, P. K. Behera, and B. N. B. Ray. (2016). Modified round robin algorithm for resource allocation in cloud computing. *Procedia Computer Science*. 85. pp. 878-890.
2. J. Bajo, F. De la Prieta, J. M. Corchado, and S. Rodríguez. (2016). A low-level resource allocation in an agent-based Cloud Computing platform. Applied Soft Computing, *48*. pp. 716-728.
3. J. Zhang, N. Xie, X. Zhang, and W. Li. (2018). An online auction mechanism for cloud computing resource allocation and pricing based on user evaluation and cost. *Future Generation Computer Systems*, *89*. Pp. 286-299.
4. Y. Sun, J. White, S. Eade, and D. C. Schmidt. (2016). ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization. *Journal of Systems and Software*, *116*. pp. 146-161.
5. S. A. Tafsiri, and S. Yousefi. (2018). Combinatorial double auction-based resource allocation mechanism in cloud computing market. *Journal of Systems and Software*. *137*. pp. 322-334.
6. H. Zheng, Y. Feng, and J. Tan. A hybrid energy-aware resource allocation approach in cloud manufacturing environment. *IEEE Access 5*. pp. 12648-12656.
7. D. Liu, X. Sui, L. Li, Z. Jiang, H. Wang, Z. Zhang, and Y. Zeng. (2018). A cloud service adaptive framework based on reliable resource allocation. *Future Generation Computer Systems*. *89*. pp. 455-463.
8. G. Xu, J. Pang, and X. Fu. (2013). A load balancing model based on cloud partitioning for the public cloud. *Tsinghua Science and Technology, 18(1)*. pp. 34-39.
9. A. Kaur, B. Kaur, and Dheerendra Singh. (2019). Meta-heuristic based framework for workflow load balancing in cloud environment. *International Journal of Information Technology*. *11(1)*. pp. 119-125.
10. N. Leontiou, D. Dechouniotis, S. Denazis, and S. Papavassiliou, (2018). A hierarchical control framework of load balancing and resource allocation of cloud computing services. *Computers & Electrical Engineering*, *67*. pp. 235-251.
11. F. Ramezani, J. Lu, and F. K. Hussain, Task-based system load balancing in cloud computing using particle swarm optimization. *International journal of parallel programming*. 42(5). pp. 739-754.
12. K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam. (2013). A genetic algorithm (ga) based load balancing strategy for cloud computing. *Procedia Technology*, *10*. pp. 340-347.
13. L. Wang, Z. Wang, S. Hu, and L. Liu. (2013). Ant colony optimization for task allocation in multi-agent systems. *China Communications*, *10(3)*. pp. 125-132.
14. X. Liu, X. Zhang, W. Li, and X. Zhang. (2017). Swarm optimization algorithms applied to multi-resource fair allocation in heterogeneous cloud computing systems. *Computing*, *99(12)*. 1231-1255.
15. V. Polepally, and K. S. Chatrapati, (2017). Dragonfly optimization and constraint measure-based load balancing in cloud computing. *Cluster Computing*, pp.1-13.
16. K. R. Babu, and P. Samuel. (2016). Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud. *Innovations in bio-inspired computing and applications*. Springer, Cham, pp. 67-78, 2016.
17. M. Lawanyashri, B. Balusamy, and S. Subha. (2017). Energy-aware hybrid fruitfly optimization for load balancing in cloud environments for EHR applications. *Informatics in Medicine Unlocked*. *8*. pp. 42-50.
18. T. Jena, and J. R. Mohanty. (2018). GA-based customer-conscious resource allocation and task scheduling in multi-cloud computing. *Arabian Journal for Science and Engineering*. *43(8)*. pp. 4115-4130.

19. P. Durgadevi, and S. Srinivasan. (2018). Resource Allocation in Cloud Computing Using SFLA and Cuckoo Search Hybridization. *International Journal of Parallel Programming*. pp. 1-17.
20. L. Kong, J. P. B. Mapetu, and Z. Chen. (2019). Heuristic Load Balancing Based Zero Imbalance Mechanism in Cloud Computing. *Journal of Grid Computing*. pp. 1-26.
21. J. P. B. Mapetu, Z. Chen, and L. Kong. (2019). Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing. *Applied Intelligence*. pp. 1-23.

AUTHORS PROFILE

Usha Kirana S P is currently Visvesvaraya Technological University (VTU), Belagavi and working in the area cloud computing and optimization. She has received her M.Tech from VTU Belagavi, in 2012 and B.E. in 2009.

Demian Antony D'Mello is a professor at Canara College of Engineering, VTU, Bantwal, India. He has published over sixty research papers in national and international journals/conferences and supervised more than 30 projects/dissertation of PG students. He has supervised more than 10 Ph.D. in the area of Web Technology, Web Services, Cloud Computing, Big Data Analytics, Distributed Computing.