Computer Systems Science & Engineering
DOI: 10.32604/csse.2023.024339
Article

Tech Science Press

An Advanced Dynamic Scheduling for Achieving Optimal Resource Allocation

R. Prabhu[1,*] and S. Rajesh[2]

[1]Department of Computer Science and Engineering, AAA College of Engineering and Technology, Sivakasi, Tamilnadu, India
[2]Department of Information Technology, Mepco Schlenk Engineering College Sivakasi, Tamilnadu India
*Corresponding Author: R. Prabhu. Email: prabhuaaacet1@gmail.com
Received: 14 October 2021; Accepted: 22 December 2021

Abstract: Cloud computing distributes task-parallel among the various resources. Applications with self-service supported and on-demand service have rapid growth. For these applications, cloud computing allocates the resources dynamically via the internet according to user requirements. Proper resource allocation is vital for fulfilling user requirements. In contrast, improper resource allocations result to load imbalance, which leads to severe service issues. The cloud resources implement internet-connected devices using the protocols for storing, communicating, and computations. The extensive needs and lack of optimal resource allocating scheme make cloud computing more complex. This paper proposes an NMDS (Network Manager based Dynamic Scheduling) for achieving a prominent resource allocation scheme for the users. The proposed system mainly focuses on dimensionality problems, where the conventional methods fail to address them. The proposed system introduced three –threshold mode of task based on its size STT, MTT, LTT (small, medium, large task thresholding). Along with it, task merging enables minimum energy consumption and response time. The proposed NMDS is compared with the existing Energy-efficient Dynamic Scheduling scheme (EDS) and Decentralized Virtual Machine Migration (DVM). With a Network Manager-based Dynamic Scheduling, the proposed model achieves excellence in resource allocation compared to the other existing models. The obtained results shows the proposed system effectively allocate the resources and achieves about 94% of energy efficient than the other models. The evaluation metrics taken for comparison are energy consumption, mean response time, percentage of resource utilization, and migration.

Keywords: Cloud computing; resource allocation; load balance; dynamic scheduling; dimensionality reduction

1 Introduction

Cloud computing is a modern technology that has massive growth in recent years. The online computing resources provide on-demand customer-oriented services effectively. In cloud computing, Virtual Machines (VM's) are an essential component that enables handling customer requests, eventually without maximizing the physical infrastructures. Day by day, cloud usage and needs increases rapidly in the same manner cloud

issues also in a noticeable growth. The cloud users submit multiple tasks through several applications, which requires stabilization on storage, computing, and memory capacity [1,2]. Load balancing is the art of balancing the workload by redistributing to the cloud computing without any, under-loaded, overloaded or idle [3,4]. In this situation, the existing traditional methods [5,6] are not effective in handling cloud computing issues, which are change over time. Among which the most important and targeted issues is adequate resource allocations. Generally, cloud computing services shown in Fig. 1 are provided under;

- IaaS (Infrastructure as a service)
- PaaS (Platform as a service)
- SaaS (Software as a service)

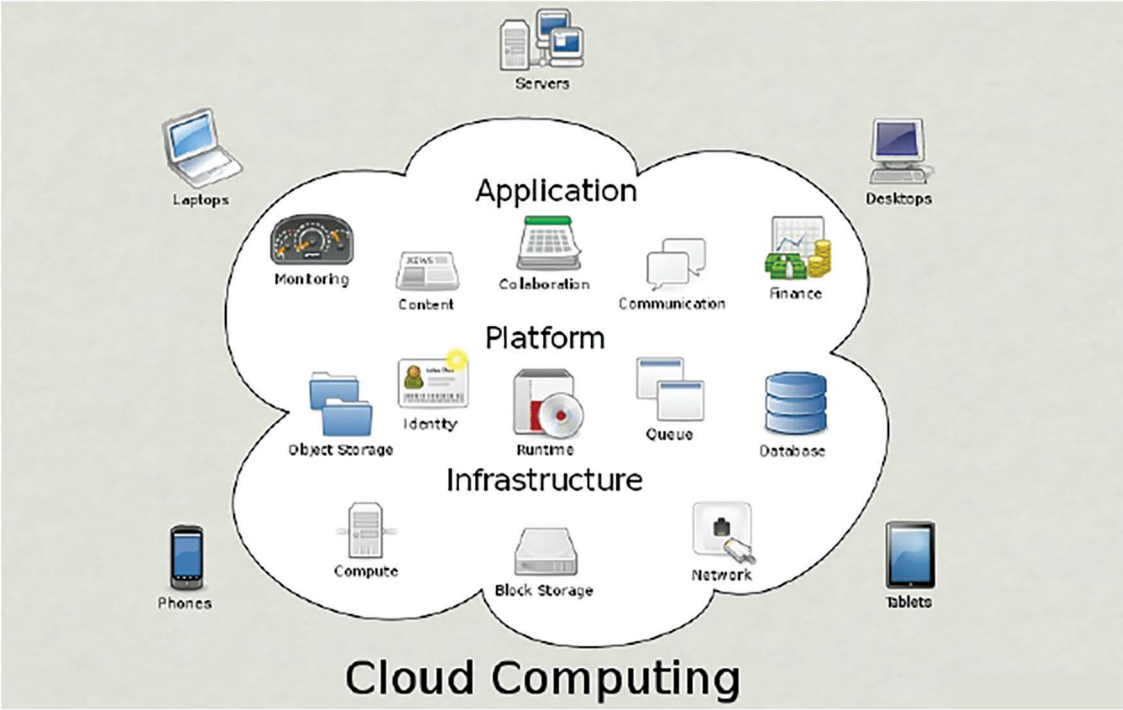

Figure 1: Cloud computing

The users avail themselves of the cloud services through the cloud service providers. The user sent their request to the service provider via the internet. Based on the obtained request, the service providers allocate the resources to the users. For effective resource allocation, resource and scheduling algorithm is followed for regulating the cloud services. Resource Allocation (RA) is allocating the available resources to the needs over the internet. The improper management of RA suffers heavily within the limit of the cloud environment [7,8]. This paper is organized as follows; Section 1 contains the introduction, Section 2 contains related works, Section 3 includes the proposed work, Section 4 contains the result section, and Section 5 contains the conclusion.

Objectives:

- Addressing the dimensionality problems, where the conventional methods fail to address them.
- Implementing three –threshold mode of task based on its size STT, MTT, LTT (small, medium, large task thresholding).
- Achieving minimum energy consumption and response time using the task merging concept

2 Related Works

Abdul-Rahman et al. (2019) [8] addressed the high-performance issue in the virtualized cloud data centers (CDCs). As cloud users are maximized widely, it is necessary to improve resource utilization maximization, especially in task scheduling. The author enhances the energy efficiency of CDC using an energy-efficient dynamic scheduling scheme (EDS). The EDS mechanism is developed for handling real-time task handling in the virtual CDCs. This mechanism applies a historical scheduling record for analyzing the VMs energy level and allocating the virtual machines' heterogeneous tasks accordingly.

Additionally, merging of same kind tasks minimize the energy and time consumptions effectively. Marahatta et al. (2013) [9] discussed the difficulties in handling the incoming loads. The significant varying of loads makes the managing and allocation of VMs more critical. A decentralized virtual machine migration approach is developed to overcome this issue and improve the quality of services. This decentralized mechanism analyzes the load information, load vectors, VM details, destination and applies a two-threshold decentralized migration algorithm. The implementation of a two-threshold decentralized migration algorithm minimizes the energy consumptions and enhances the quality of service. Wang et al. (2019) [10] discussed the various load unbalancing situations. The author evaluates the undesirable facts that occur during facets-overloading and under-loading in the cloud. No earlier works describe the need for load balancing methods, especially with inclusive, extensive, organized, and hierarchical classifications. The author addressed these areas effectively with a comprehensive encyclopedic review on several load balancing algorithms. Afzal et al. (2020) [11] proposed a stochastic approximation method for handling foresighted task scheduling. The Proposed Stochastic Optimization-Based Solution design a policy that reduces task responding times. The task time may vary due to its nature, and hence it is difficult to estimate the task length using the rule-based or threshold-based schemes. In this situation, the proposed model-free reinforcement learning solution contains self-adaptive and self-learning capabilities, which calculates the virtual machine's queue lengths dynamically. This works effectively on the unsteady requests, which maximize the makespan and improvises the overall system's performance. Mostafavi et al. (2020) [12] addressed the load imbalance issues where the Conventional Q-learning model are failed to overcome. The author proposes a fair-based reinforcement learning model for effectively handling resource management. According to the proposed work, rule-based resource managers play a vital role in calculating job delays and deadlines. Using this mechanism, the author effectively handled the dimensionality problems during the heavy loads. The proposed system's principle is based on the input; if more input means more weight. The maximum weights determine the input's importance, and the annealing steps are used to make the optimal decisions. Karthiban et al. (2019) [13] introduce Cost-based job scheduling (CJS) algorithm to schedule the tasks effectively. The existing works are fine with small loads, but Scheduling becomes complex while facing huge loads. The proposed CJS is intensely concentrated in data transfer costs, network costs, and computation costs. Based on the three costs, the total cost is determined. CJS takes the scheduling decision based on the job characteristics, data, and computation requirements. The obtained results achieve maximum lifespan comparing to the other existing works. Mansouri et al. (2018) [14] addressed the load balancing issue using the probability theory. The probabilistic load balancing (PLB) approach works with two phases. In the first phase, the PLB sorts the non-decreasing task and identifies the available VM's. In the second phase, based on the non-decreasing task, the VMs are allocated. Additionally, the proposed scheme notes the task arrival task and sorts the tasks accordingly.

Panda et al. (2019) [15], conducted a brief study on cloud computing-basic terminologies in all aspects which includes mobile cloud computing, cloud security, cloud models, resource allocation and simulators. Krishnaraj et al. (2019) [16], described a deep analysis and survey about the existing data placement methods. Because data placements are the major impact which has direct impact on workflows execution cost, performances and time. c. Kaur et al. (2019) [17], discussed about the on-demand auto-scaling of

the resources which has dynamic workloads are very complex. It impacts the QoS and an existing challenge for the web applications running in the cloud [18]. In this work the authors conducted a survey about various auto-scaling techniques in cloud computing. Nayyar et al. (2011) [19], addressed collaborative adaptive scheduling and virtual resources problem on cloud workflow tasks. Based on which they proposed a fine-grained cloud computing system model novel collaborative adaptive scheduling mechanism using multi-agent society and reinforcement learning systems. Jain et al. (2020) [20], proposed Registration Authentication Storage Data Access (RASD) scheme for enabling security on cloud catalogs. RASD is a stage by stage process which involves client secret key authentication, cloud registry information access and Enlistment of clients.

3 Proposed System

3.1 Network Manager Based Dynamic Scheduling (NMDS)

Our proposed NMDS (Network Manager based Dynamic Scheduling) is developed with Network Manager (NM), which plays an essential role in the entire system. From the process begins with network deployment till in successful communication, NM took responsibility for managing, controlling, and monitoring the whole mechanism. The main motto of implementing NMDS is achieving quality of services (QoS). Earlier systems fail in managing the dynamic loads because they are of different types and different sizes. Another instance is the task can be received from various applications from anywhere and anytime various users. The NMDS applies NM (network manager) as a central resource for achieving the Scheduling dynamically without any network performance issues. The proposed system is dynamic, enabling automatic migration whenever any VM fails or busy and makes the communication more stable. The four main parameters in NMDS are listed below;

1. System Model
2. Resource Allocation and Monitor
3. Task Model
4. Task Migration

3.1.1 System Model

The NMDS receives the tasks from various multiple users and arranges the task in a queue. The received task is of different types and sizes. Initially, NMDS counts the total task received and organized in the queue and then analyzes the task nature. Based on the user's file size, the task is categorized into three kinds such as small, medium, and large. The task categorization is based on its size, which depends on the kb and MB. The small task contains text, word, pdf files which are smaller in size and can execute much quicker than the others. The medium type task includes the file types with images (HD images themes and wallpapers and HD audio) files. The size of the medium type task is in MB and GB's. The last type is the large task, including audio and video files (HD audio, video, and blue-ray). The large task needs high computing energy and time for executions.

The above Fig. 2 illustrates the proposed architecture; the NM collects the task from various users such as user 1, user 2, user 3, user…n. The user can submit the task from anywhere and of any type. VM's incoming loads are from various applications; hence its size and types are undetermined. Most of the existing systems not effective in managing the task of multiple variants. In the proposed method, NM took the responsibility of counting the task and categorization.

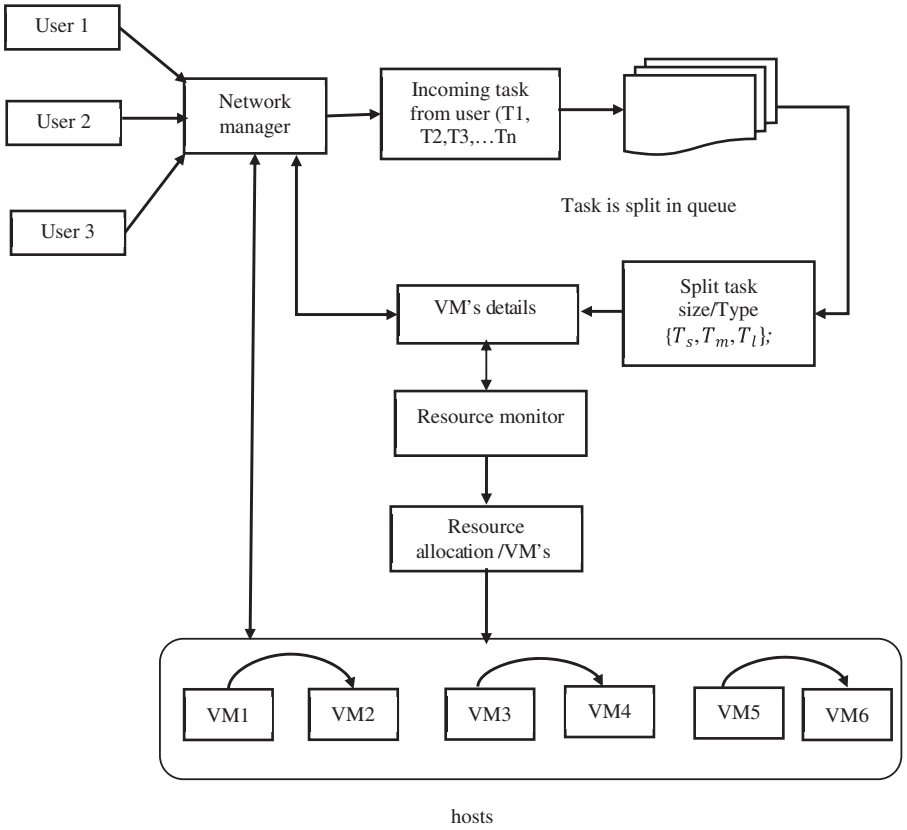

Figure 2: Proposed architecture

3.1.2 Resource Model

In this section, the resource model is described, which contains the VM details and its implementation briefly. The VMs are the resources, and those in an active state can participate in the execution. The VM is placed in the host for functioning dynamically. The VM of one host can be allocated to another host also. The VM dynamic mechanism is explained in the task and VM model in detail. Let's discuss the VM details and its capacity level to perform the tasks. Generally, VM details include a processor, memory size, disk storage, and bandwidth. Based on these specifications, the task execution is split into two types: tasks requiring high CPU and low memory, and a second task that needs low CPU and increased memory. The VM details, which are in execution, including both active VM and idle VM's details, are registered with the NMDS. Based on the details, NMDS knows the total available VM's and their status, such as available, busy, and idle. These statuses can change anytime because the network is dynamic. All these VM's are under the monitoring of NM, and according to its resources state, it can allocate dynamically to the tasks.

NM contains every detail about the entire mechanism such as total VMs, total busy VMs, total task completed, total task migrated, the total number of task discarded, etc. The below section describes detail about how the VM are scheduled and how migration is done effectively.

The proposed system consists of multiple hosts. The VM is executed as a single physical machine for a host, and it can be distributed to multiple hosts also. H = {H1, H2, H3… Hn} are the host which contains the VM = $Vm_{ix} \rightarrow vm_1, vm_2, vm_3, \ldots vm_n$. The total task and its task types are determined as $T_{ix} \rightarrow T_1, T_2, T_3, \ldots T_n$ and $T_{ix} \rightarrow T_{type}\ \{T_s,\ T_m,\ T_l\}$. H_{ix} (H_{active}, H_{idle}). determine the host either in the active state or idle state. The VM details such Vm_p states its processing capacity, Vm_m states VMs primary memory size, Vm_n its

network bandwidth, V_d its disk storage, etc. The other terms and descriptions utilized in the proposed work are described the below Tab. 1.

Table 1: Key terms and descriptions

T_{ix}	T_1, T_2, T_3, …T_n set of tasks
T_{type}	Task type
T_s	Small task
T_m	Medium task
T_l	Large task
T_{siz}	Text data size
N_{task}	Total number of task
v_{siz}	Video data size
A_{siz}	Audio data size
N_{mgr}	Network manager
q	Queue process
Vm_{ix}	Set of virtual machines
$vm\ (p_{ix})$	Virtual machine parameter
Vm_p	Processing capacity
Vm_m	Primary Memory size
Vm_n	Network bandwidth
V_d	Disk storage
H_{ix}	Set of hosts
$\mathrm{H(f_i\ v_i)}$	frequency, voltage is discreet pairs
T_{hold}	Threshold
STT	Small thresolding task
MTT	Medium thresholding task
LTT	Large thresolding task

3.1.3 Task and Virtual Machine Model

The task model and its implementation contain two-stage of the process, such as task allocation and VM allocations. Initially, the queue's received task are taken, which are in multiple numbers from multiple users. The total number of task collected by the NM and its queueing process is expressed as below;

$$T_{ix} \rightarrow T_1,\ T_2,\ T_3,\ \ \ldots T_n \tag{1}$$

Next, the task is categorized according to its type, such as small, medium, large size tasks.

$T_{type} \rightarrow \{T_s,\ T_m,\ T_l\}$

T_s-Defines the small task, which includes text, pdf, word files. The total size of these tasks is less than or equal to MB.

T_m-Defines the medium task, which includes audio, HD image files. The total size of these tasks is less than or equal to MB or GB.

T_l-Defines the large task, which includes video files. The total size of these tasks is equal to or greater than GB.

NM has absolute control and monitors these tasks promptly. Initially, the task received, and the categorization mentioned above is done based on the below calculations;

$$T_{type} \rightarrow \sum_{i=0}^{n} \frac{T_{siz}}{N_{task}} \tag{2}$$

T_{type}–Task Type

T_{siz}–Task Size

N_{task}–Total Number of Tasks

Next, the VM allocation, initially the VM taking part in the execution is analyzed based on the processing capacity, primary memory, network bandwidth, and disk storage. This analysis determines the VMs capacity level and which VM is allocated to which task. The VM and its parameters are expressed below;

$$N_{mgr} \leftarrow\ Vm_p,\ Vm_m,\ Vm_n,\ V_d \tag{3}$$

where, N_{mgr}–Network Manager

Vm_p–Processing Capacity

Vm_m–Primary Memory

Vm_n–Network Bandwidth

V_d–Disk Stoarge

In this proposed environment, users submit the real-time tasks independently to the service providers. During receiving the task, NM does not know anything about the task, especially its complexities and underlying infrastructures. The independent real-time task in the queue can be expressed as below;

$$T_{ix} \rightarrow T_1,\ T_2,\ T_3,\ \ldots T_n \tag{4}$$

Whereas T_1, T_2, T_3, $\ldots T_n$ are the obtained task indexed in the queue. The working and allocation of task and VM under NM's supervision is described in below algorithm.

Algorithm for task and virtual machine allocation

1. Input: user ( ); Task (t_{ix})*( ); vm (d) details ( );*
2. Begin ();
3. Set of task
4. Initialize the network manager (N_{mgr})
5. {

(Continued)

Algorithm: (continued)

6. $T_{ix} \rightarrow T_1, T_2, T_3, \ldots T_n$ *; // task count*
7. $T_{ix} \rightarrow T_{type}\ \ T_s,\ T_m,\ T_l$*;*
8. $T_s \rightarrow \sum_{i=0}^{n} \frac{T_{siz}}{N_{task}}$ *kbs // text data size and the number of text size*
9. $T_m \sum_{i=0}^{n} \frac{I_{siz}}{N_{task}}$ *mb /bs // number of image size and total number of images*
10. $T_l \sum_{i=0}^{n} \frac{V_{siz\ and\ A_{siz}}}{N_{task}}$ *gb /bs // number audio and video files, total number of audio and video files*
11. $T_{type} \rightarrow T_s, T_m, T_l$*;*
12. $N_{mgr} \leftarrow T_{type}$ *// update the network manager*
13. Create
14. Task in queuing process
15. $q \rightarrow T_{ix}$ *// task on queue process;*
16. $q \rightarrow N_{mgr}$ *// update the queue the network manager*
17. Initialize the task parameter
18. Create virtual machines
19. $Vm_{ix} \rightarrow vm_1, vm_2, vm_3, \ldots vm_n$
20. $vm\ (p_{ix}) \rightarrow Vm_p, Vm_m, Vm_n, V_d$
21. $N_{mgr} \leftarrow\ Vm_p, Vm_m, Vm_n, V_d$ *// virtual machine parameters update the network manager*
22. Create host
23. $H_{ix} \rightarrow H_1, H_2, H_3, \ldots H_n$
24. Host state
25. $H_{ix} \rightarrow H_{active}, H_{idle}$
26. $H_{ix} \rightarrow H(f_i, v_i)$ *// frequency, voltage is discreet pairs of frequency and voltage*
27. ..;
28. }
29. End

This section describes the working of dynamic Scheduling with a thresholding approach. NM is initialized first, and it prepares the VM's list, VM's arrange in descending order. For preparing the VM's list, NM analysis the VM parameters, including processing capacity, primary memory, network bandwidth, and disk storage. The task list is then prepared based on its types, such as small, medium, and large, $T_{type} \rightarrow T_s,\ T_m,\ T_l$ calculated using the file size. The task and its threshold value determine which task is allocated to which available VM. It is calculated under three instances: task size with the total number of tasks less than 25% *is* $T_s \rightarrow \sum_{i=0}^{n} \frac{T_{siz}}{N_{task}} > 25\%$ labeled as STT (Small Task Thresholding). The VM's allocated for STT equal or more capacity to the STT. The second case handling of the medium task, *the* task size with the total number of tasks between above 25% to 65%, is $T_m \sum_{i=0}^{n} \frac{I_{siz}}{N_{task}} > 65\%$ labeled as MTT (Medium Task Thresholding). The VM which has higher capacity level the VM's allocated to STT are allocated to MTT. Finally, VM allocation for the large task can be determined by the task size, with the total number of tasks above 65% is $T_l \sum_{i=0}^{n} \frac{V_{siz\ and\ A_{siz}}}{N_{task}} < 65\%$ labeled as LTT (Large Task

Thresholding). The VM list with the high capacity is allocated for handling the large task. The VM parameters determine the VM capacity level, explained in detail in the below sections. This thresholding-based task categorization is updated to the NM. The working mechanism of dynamic Scheduling with a thresholding is determined in the below algorithm.

Algorithm dynamic scheduling with thresholding

1. Input: set of task T_{ix}; *set of virtual machine* Vm_{ix}; *set of host* H_{ix};
2. Begin ( );
3. Initialize the network manager (N_{mgr})
4. {
5. Process { }
6. Order the Vm list descending order Vm_{list}( ) ;
7. $Vm_{list} \rightarrow$ *vm* (p_{ix}) ;
8. Assign the task $T_{type} \rightarrow T_s$, T_m, T_l *// assign the task type wise;*
9. Assign the $T_{hold} \rightarrow Threshold(Vm_{list})$;
10. }
11. Create Vm's based on T_s
12. Case (i) small task thresholding (STT)

13. $T_s \rightarrow \sum_{i=0}^{n} \frac{T_{siz}}{N_{task}} > 25\%$

14. Allocate the vm's vm (p_{ix}) > 25%
15. Case (ii) medium task thresholding (MTT)

16. $T_m \sum_{i=0}^{n} \frac{I_{siz}}{N_{task}} > 65\%$

17. Allocate the vm's vm (p_{ix}) < 65%
18. Case (iii) large task thresholding (LTT)

19. $T_l \sum_{i=0}^{n} \frac{V_{siz} \text{ and } A_{siz}}{N_{task}} < 65\%$

20. Allocate the vm's vm (p_{ix}) < 65%
21. ($N_{mgr} \leftarrow vm_{list}$, T_{type}, T_{hold}, *STT*, *MTT*, *LTT*);
22. Update the network manager;
23. End process
24. End

3.1.4 Migration Model

The proposed system is developed for large-scale data centers, and most of the existing system contains a central resource manager for task migration. The central resource manager takes the appropriate decisions when any VM fails or in busy. But the major issue is the central resource manager is subject to a single-point failure. On this occasion, it causes total damage to the entire system. In our proposed approach, a decentralized VM system is implemented under the supervision of NM. During the decentralized VM deployment, each VM registers their details with the NM, and hence NM knows each VM capacity level from the beginning. In this process, VM's need to manage the varying workloads hosted by the users

from different applications. Hence, the CPU utilization of the physical nodes will change a lot over time. When the CPU utilization is too high, some VMs should be migrated to others to release some resource capacity exceeding a certain level. On the other hand, if the CPU utilization is too low, below a certain level, we regard the node as "underutilized", and the NM migrates the task to the next available VM. NM maximum possibility of VM migration is with the LTT. The VM allocated for executing the STT can be done successfully, which does not need any migration because the task file size is small to execute without any failure. NM migrates the VM for MTT whenever it's required but most frequently while executing LTT. Because VM executing LTT needs to have a high capacity, sometimes the current VM may get overloaded. On this occasion, the NM migrates the task to the next available, which has a higher energy level. Above Fig. 2 illustrates the active VM s in the network indexing to another active VM, which means one VM is busy or overloaded the NM migrated the task to the next available VM. The working of task migration is described in the below algorithm 3 and Fig. 3.

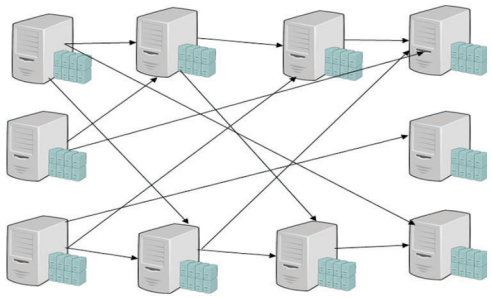

Figure 3: Decentralized VM system

Algorithm for task migration

1. Procedure Task migration ( );
2. Input: list of all type of tasks
3. Initialize the network manager (N_{mgr})
4. {
5. Initialize the [Vm List]
6. Initial vm list [ ] → 0;
7. $T_{type} \rightarrow$ [*STT*, *MTT*, *LTT*] ;
8. $N_{mgr} = \frac{\sum_{i=0}^{n} Vm(p_{ix}) \rightarrow (Vm_p, Vm_m, Vm_n, V_d);}{T_t(T_{type})}$
9. For each [*vm list*] ;
10. {
11. ($Vm(p_{ix}) > T_{type}$); *// STT, MTT task type*
12. Vm's are available update the vm list [ ];
13. Migration status [ ];
14. (Vm to mig == NULL);
15. Update the vm list [ ]→ N_{mgr};
16. Else if (Vm to mig)
17. ($Vm(p_{ix}) > T_{type}$); *MTT, LTT;*
18. For each [*vm in this. vm list*] ;

(Continued)

Algorithm: (continued)

19. {
20. If (vm to be migrated) continue;
21. (Vm to mig == available);
22. Update the vm list [ ]→ N_{mgr};
23. break; //find a VM to migrate
24. }
25. }
26. if (vmToMig==null) break; //case (c)
27. find Des t (vmToMig); //find a destination for the current VM to migrate to
28. }

The migration model is processed as below;

Step 1: NM is initialized, and it begins the process with the VM list and task list. The task type and its threshold levels, such as STT, MTT, and LTT, are discussed in Section 3.1.3.

Step 2: Based on the task type, NM allocates the available VM. The Task execution begins with pairing a task such as STT, MTT and then MTT, LTT. For the STT, MTT no migration is required because the task size is small and hence the migration status is null; it is updated to the NM.

Step 3: Next handling of task with MTT, LTT, which requires higher system capacity. NM checks the VM list and allocates the higher level VM to the LTT. Due to the dynamic workload, some VM gets overutilized on dealing with LTT and updated to the NM. Then NM finds the next available higher-level VM in the list and migrates the task to that next higher-level VM. This migration is continuing until the destination is reached. At all the stage, VM migration status is updated to the VM.

4 Results and Discussion

Experimental work is carried out to evaluate the overall performance of our proposed NMDS. A comparison analysis is carried out with the existing Energy-efficient Dynamic Scheduling scheme (EDS) and Decentralized Virtual Machine Migration (DVM). The comparison's performance metric is total energy consumption, mean response time, percentage of resource utilization, and migration. The observation is conducted in the simulation environment of CloudSim. Powered by CloudSim toolkit. The system parameters which includes CPU (Intel(R) Core(TM) i5-3230 M, 2.60 GHz), Memory (8.0 GB), Hard Disk (750 GB), Windows 10 operating system, NetBeansIDE, JDK 8.0.

Performance metrics:

Tab. 2 states each VMs type CPU and memory capacity. Tab. 3 displays each task type's CPU and memory capacity. These two are the essential elements where network bandwidth and hard disk capacity are not considered critical because they can easily match all tasks. Tab. 4 describes the simulation setup's host specifications, and here all simulated tasks are dynamic and heterogeneous. In this simulation, the host CPU core performs about 1500 MIPS, 2000 MIPS, and 2800 MIPS.

Table 2: VM types with parameter

VM type	CPU (GHZ)	RAM size (GB)
Type 1	0.10	0.018
Type 2	0.15	0.06

(Continued)

Table 2 (continued)

VM type	CPU (GHZ)	RAM size (GB)
Type 3	0.08	0.009
Type 4	0.19	0.012
Type 5	0.15	0.015
Type 6	0.12	0.013
Type 7	0.08	0.05
Type 8	0.05	0.0.37

Table 3: Task types with parameter

Task types	CPU (GHZ)	RAM size (GB)
STT	0.01–0.09	0.003–0.016
MTT	0.02–0.11	0.004–0.020
LTT	0.05.−0.20	0.005–0.030
STT	0.01–0.05	0.002–0.014
MTT	0.03–0.14	0.005–0.025
LTT	0.023–0.29	0.006–0.035
MTT	0.08–0.12	0.005–0.0.15
LTT	0.09–0.18	0.006–0.018

Table 4: Simulated host specification

Host types	CPU (MIPS)	Host count
Type-1	1500	3500
Type-2	2000	3000
Type-3	2800	3800

The performance metrics are;

1) **Total energy Consumption**: The total energy consumed for completing a task within the deadline.
2) **Response Time:** Optimal time is taken to respond to the user.
3) **Resource Utilization:** Total number of resources utilized for scheduling the task.
4) **Number of migrations:** Total number of migrations for completing a specific task type.

Fig. 4 shows the total energy consumption obtained by each algorithm concerning the task count. The X-axis determines the task count, and the Y-axis defines the total energy consumed. The respective task counts are 2000, 4000, 6000, 8000, 10000. In all the instances, the energy consumed by the proposed NMDS is very minimal in comparison to the EDS and DVM. The reason for this is our proposed NMDS

applied task emerging on similar task types, which minimize the energy and time. Additionally, categorizing the task type and scheduling the VM according to the task effectively achieves optimal resource utilization.

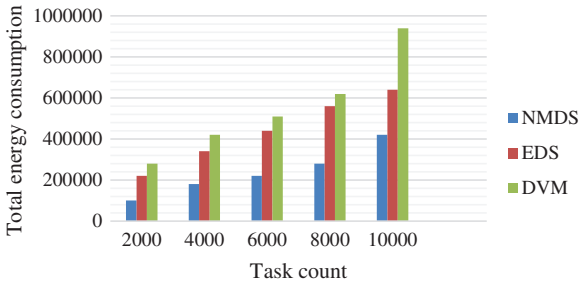

Figure 4: Total energy consumption *vs.* task count

Fig. 5 illustrates the mean response time achieved by each algorithm respective to the task counts. At all the task counts, NMDS responds to the user with minimum time than the EDS and DVM. It is because the NMDS first analyzes the task count and prioritizes the task with its deadline. The categorization concept in NMDS, such as small, medium, and large with task migration, enables NMDS to complete the task and respond effectively to the users quickly.

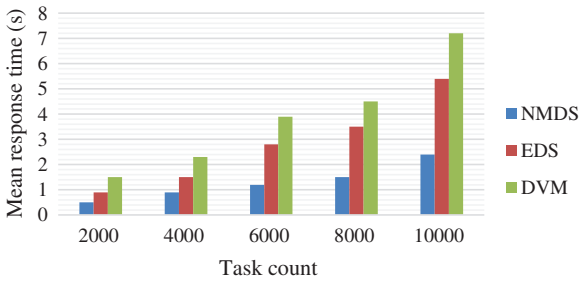

Figure 5: Mean response time *vs.* task count

Fig. 6 describes the total amount of resources utilized for the task. In which the Y-axis states the resource used in percentage, and X-axis displays the task count. The task count taken for observation is 2000, 4000, 6000, 8000, and 10000, respectively. In the proposed NMDS, the host with the active and idle state initially notes, and the NM monitors each host effectively. As the task is entered, the NM allocates the available VM according to the task. During the new task entered or migration occurred, the idle state's host is changing to an active state and allocated with the task. Hence it is achieving optimal resource utilization. The above graph with obtained results also proves the proposed NMDS resource utilization percentage is higher than the EDS and DVM.

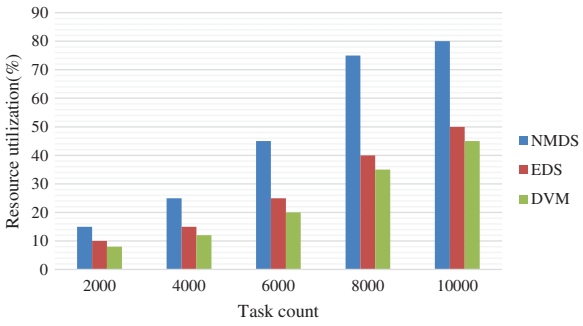

Figure 6: Resource utilization *vs.* task count

Fig. 7 describes the proposed VM migration with small, medium, and large utilization thresholds. The above graph shows that a lower threshold maximizes the VM migrations. At the lower threshold, most VM considered themselves as "underutilized" and enabled multiple migrations to achieve optimal resource utilization. Sometimes, large thresholds have minimum migration. This is because VMs are in demand, and migration will occur only when the current VM gets overutilization. The threshold types taken for the observation are LTT–Large Task Type, MTT–Medium Task Type, and STT–Small Task Type. The X-axis states the thresholding type, and Y-axis displays the number of migration occurred with that type. The above graph shows the number of migration is maximum with STT in comparison to MTT and LTT.

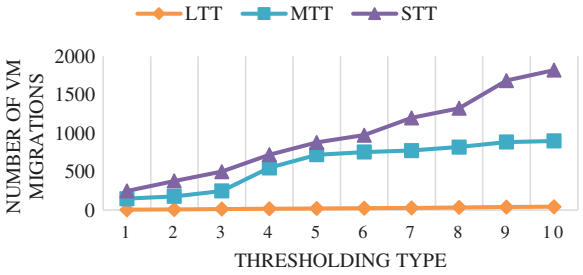

Figure 7: Number of migration *vs.* threshold type

5 Conclusion

In cloud data centers, the most critical issues are energy consumptions and lack of resource utilization. Most of the existing methods are failed to handle improper resource allocations which results to load imbalance and other service related issues. Another important issues need to consider is maximization of energy consumption due to lack of optimal resource allocating scheme. In this paper, we propose NMDS, which achieves dynamic Scheduling using Network Manager. The NM counts arrived and classified the arrived task into three mediums based on the threshold values. The decentralized VM migration approach effectively balances the loads and VM failures. The task migration enables quick response time by merging and executing similar task types. These features enhance the overall resource utilization, energy, and time consumption. To prove the performance of the proposed NMDS, experimental work is carried, and the obtained results are compared with EDS and DVM. The performance metrics considered are total energy consumption, mean response time, percentage of resource utilization, and migration. The task merging enables minimum energy consumption and response time very efficient and achieves about 94% of improvement in minimum energy consumptions. On all the metrics, the proposed NMDS is far better than the EDS and DVM. In future, the proposed mechanism can be extended to achieve security because security breach is considered as another major issues next to energy consumption.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Dam, G. Mandal, K. Dasgupta and P. Dutta, "Genetic algorithm and gravitational emulation based hybrid load balancing strategy in cloud computing, " in *Proc. of the 2015 Third Int. Conf. on Computer, Communication, Control and Information Technology (C3IT)*, Hooghly, India, pp. 1–7, 2015.

[2] A. Dave, B. Patel and G. Bhatt, "Load balancing in cloud computing using optimization techniques: A study," in *Proc. Int. Conf. on Communication and Electronics Systems (ICCES)*, Coimbatore, India, pp. 1–6, 2016.

[3] R. Achar, P. S. Thilagam, N. Soans, P. V. Vikyath, S. Rao *et al.,* "Load balancing in cloud based on live migration of virtual machines," in *Proc. Annual IEEE India Conf. (INDICON)*, India, pp. 1–5, 2013.

[4] D. Magalhaes, R. N. Calheiros, R. Buyya and D. G. Gomes, "Workload modeling for resource usage analysis and simulation in cloud computing," *Computers & Electrical Engineering*, vol. 47, pp. 69–81, 2015.

[5] H. Faragardi, A. Rajabi, K. Sandstrom and T. Nolte, "EAICA: An energy-aware resource provisioning algorithm for real-time cloud services," in *Proc. IEEE 21st Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, Berlin, Germany, pp. 1–10, 2016.

[6] S. K. Mishra, D. Puthal, B. Sahoo, P. P. Jayaraman, S. Jun *et al.,* "Energy-efficient VM-placement in cloud data center," *Sustainable Computing: Informatics and Systems*, vol. 20, pp. 48–55, 2018.

[7] M. Stillwell, D. Schanzenbach, F. Vivien and H. Casanova, "Resource allocation using virtual clusters," in *Proc. 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, DC, United States, pp. 260–266, 2009.

[8] O. Abdul-Rahman, M. Munetomo and K. Akama, "Live migration-based resource managers for virtualized environments: A survey," in *Proc. of the 1st Int. Conf. on Cloud Computing, GRIDs, and Virtualization (Cloud Computing '10)*, Libson, Portaugal, pp. 32–40, 2010.

[9] A. Marahatta, S. Pirbhulal, F. Zhang, R. M. Parizi, K. R. Choo *et al.,* "Classification-based and energy-efficient dynamic task scheduling scheme for virtualized cloud data center," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1376–1390, 2021.

[10] X. Wang, X. Liu, L. Fan and X. Jia, "A decentralized virtual machine migration approach of data centers for cloud computing," *Mathematical Problems in Engineering*, vol. 2013, no. 878542, pp. 1–10, 2013.

[11] S. Afzal and G. Kavitha, "Load balancing in cloud computing –A hierarchical taxonomical classification," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 8, no. 22, pp. 1–18, 2019.

[12] S. Mostafavi and V. Hakami, "A stochastic approximation approach for foresighted task scheduling in cloud computing," *Wireless Personal Communications*, vol. 114, pp. 901–925, 2020.

[13] K. Karthiban and J. S. Raj, "An efficient green computing fair resource allocation in cloud computing using modified deep reinforcement learning algorithm," *Soft Computing*, vol. 24, pp. 14933–14942, 2020.

[14] N. Mansouri and M. M. Javidi, "Cost-based job scheduling strategy in cloud computing environments," *Distributed and Parallel Databases*, vol. 38, pp. 365–400, 2020.

[15] S. K. Panda and P. K. Jana, "Load balanced task scheduling for cloud computing: A probabilistic approach," *Knowledge and Information Systems*, vol. 61, pp. 1607–1631, 2019.

[16] N. Krishnaraj, M. Elhoseny, E. L. Lydia, K. Shankar and O. ALDabbas. "An efficient radix trie-based semantic visual indexing model for large-scale image retrieval in cloud environment," *Software: Practice and Experience*, vol. 51, no. 3, pp. 489–502, 2021.

[17] A. Kaur, P. Gupta, M. Singh and A. Nayyar, "Data placement in aera of cloud computing: A survey, taxonomy and open research issues," *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 377–398, 2014.

[18] P. Singh, P. Gupta, K. Jyoti and A. Nayyar, "Research on auto-scaling of web applications in cloud: Survey, trends and future directions," *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 399–432, 2019.

[19] A. Nayyar, "Interoperability of cloud computing with web services," *International Journal of Electro Computational World & Knowledge Interface*, vol. 1, no. 1, pp. 1–12, 2011.

[20] R. Jain and A. Nayyar, "A novel homomorphic RASD framework for secured data access and storage in cloud computing," *Open Computer Science*, vol. 10, no. 1, pp. 431–443, 2020.