International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-12 Issue-5, June 2023

Modeling and Simulation of Real-Time Virtual Machine Allocation in a Cloud Data Center

S. Jason

***Abstract:** For dynamic resource scheduling in cloud data centers, a novel lightweight simulation system is proposed; two existing simulation systems at the application level for cloud computing are reviewed; and results gained using the suggested simulation system are examined and discussed. The utilisation of resources and energy efficiency in cloud data centres can be enhanced by load balancing and the consolidation of virtual machines. An aspect of dynamic virtual machine consolidation that directly affects resource usage and the quality of service the system is delivering is the timing of when it is ideal to reallocate Virtual Machines from an overloaded host [1]. Because server overloads result in a lack of resources and a decline in application performance, they hurt the quality of service. To determine the best answer, existing approaches to host overload detection typically rely on statistical analysis inspired by natural phenomena. The drawbacks of these strategies include providing less-than-ideal outcomes and preventing the explicit articulation of a Quality of Service target. By optimizing the mean inter-migration time under the defined Quality of Service target, ideally, we present a novel method for detecting host overload for any stationary workload that is known and a particular state configuration [2]. We demonstrate that our technique surpasses the best benchmark algorithm and achieves over 88% of the performance of the ideal offline algorithm through simulations using real-world workload traces from more than a thousand Virtual Machines.*

***Keywords:** Cloud Computing; Data Centres; Dynamic Resource Scheduling; Lightweight Simulation System*

I. INTRODUCTION

The emergence of cloud computing is based on several recent developments in virtualisation, grid computing, web computing, utility computing, and related fields. Through the internet or intranet, cloud computing offers both platforms and applications on demand [1]. The concealment and abstraction of complexity, the efficient utilisation of remote resources, and the virtualisation of resources are among the primary advantages of cloud computing. The Google App Engine [2], the IBM Blue Cloud [3], Amazon EC2 [4], and Microsoft Azure [5] are a few examples of new cloud computing platforms. Software, computational, and storage network resources can be shared, allocated, and aggregated on demand using cloud computing. As there are still many complex problems to be solved, cloud computing is still seen as being in its infancy [1,6,7,8].

Manuscript received on 18 May 2023 | Revised Manuscript received on 26 May 2023 | Manuscript Accepted on 15 June 2023 | Manuscript published on 30 June 2023.
*Correspondence Author(s)
S. Jason*, Department of Information Technology, AUCA University, Kigali 2461, Rwanda. Email: sebagenzij@gmail.com, ORCID ID: https://orcid.org/0000-0002-0073-4691

Figure 1 illustrates the complete ontology developed by Youseff et al. [9] for segmenting the cloud into five primary levels from top to bottom:

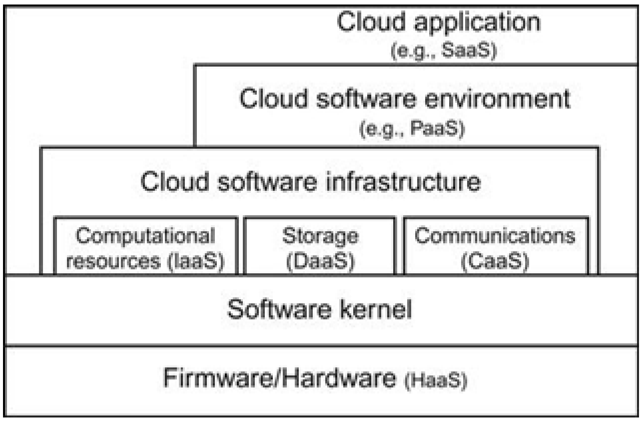

Figure 1. Layered Architecture of Cloud Computing [9].

The relationships and interdependencies with earlier technologies are also depicted in Figure 1. This study focuses on infrastructure as a service (IaaS) in cloud data centres. A distributed network with multiple computational nodes (such as servers), storage nodes, and network devices can form the structure of a CDC. A variety of resources, including CPU, memory, network bandwidth, etc., are used to create each node. Each resource has matching properties that go with it. For cloud service providers, there are many distinct kinds of resources. This essay concentrates on IaaS. This paper's definition and model are intended to be sufficiently comprehensive for use by a range of cloud service providers. Applications in a traditional data centre are connected to specific physical servers, which are often over-provisioned to handle workload spikes and unplanned outages. Due to wasted energy and floor space, low resource efficiency, and high administration overhead, such layout rigidity makes data centres expensive to maintain. Current CDCs can be made more adaptable, secure, and capable of on-demand allocation by using virtualisation technologies. With virtualisation, CDCs should be able to move applications without causing any disruption from one set of resources to another. With virtualisation, CDCs should be able to move applications without causing any disruption from one set of resources to another. The technique of resource scheduling is crucial to CDCs. Algorithms for scheduling have been the subject of extensive research. The majority of them are used for server farms or traditional web servers' load balancing. Considering the allocation and migration of reconfigurable virtual machines (VMs) and the integration of features for hosting physical machines is one of the challenging scheduling problems in CDCs.

New algorithms approach CPU, memory, and network bandwidth as an integrated system for both PMs and VMs, in contrast to classic load balancing scheduling algorithms that only consider physical servers with a single factor (such as CPU). Real-time VM allocation for numerous simultaneous processes and PMs is also considered.

The CDC's size and density increased in tandem with the challenges that arose from the growth of cloud computing. These issues include, for instance, how to dynamically and intensively manage physical and virtual resources, how to increase elasticity and flexibility (which can enhance service while lowering costs and risk management), and how to assist clients in creating flexible, dynamic, and adaptive infrastructure that enables businesses to ensure sustainable future growth without increasing spending. Because the application developers cannot control and handle the network environment, it is very challenging to conduct extensive research on all these issues in real internet platforms. Furthermore, although network conditions are unpredictable and uncontrollable, they nevertheless have an impact on how well the solutions are evaluated. Building a data centre simulation system that offers visualised modelling and simulation for large-scale applications in cloud infrastructure would enable the study of dynamic and large-scale distributed settings. The application workload statement, which comprises user information, data centre position, the number of users and data centres, and the quantity of resources in each data centre, can be described by a data centre simulation system. The data centre simulation system generates response requests based on this data and distributes them to VMs. Application developers can assess appropriate tactics, such as assigning reasonable data centre resources, selecting a data centre that meets specific needs, and reducing expenses, by using a data centre simulation system. The GridSim toolbox for modelling and simulating distributed resource management for grid computing was introduced by Buyya et al. [7]. Dumitrescu and Foster introduced the GridSim tool [8]. Buyya et al.'s [7] introduction of modeling and simulation of cloud computing systems at the application level included discussion and comparison of straightforward scheduling methods like time- and space-sharing. A cloud computing simulator called Cloud Sim [7] performs the following tasks:

1. Supporting both a single physical computing node and a Java VM data centre's modelling of a large-scale cloud computing architecture.
2. Data centre modelling, service provider, and scheduling and distribution tactics.
3. Supplying virtual engines that are useful for establishing and managing multiple independent and cooperative virtual services within a data centre node.

Time- and space-sharing should enable quick and simple switching between processing cores. Cloud Analyst [12] seeks to achieve the best scheduling across user groups and data centers depending on the existing configuration.

SimJava [11] and GridSim [10] are the foundations for both CloudSim and CloudAnalyst, which makes them complex. Additionally, CloudSim and Cloud Analyst only consider workloads at the application level and regard a CDC as a sizable resource pool. They may not be suitable for an IaaS simulation, where each VM is treated as a resource that must be requested and allocated.

Wood et al. [13] presented VM migration methodologies and suggested migration algorithms.

Zhang compared significant load balancing scheduling techniques for conventional web servers [15]. By taking into account both servers and storage in cloud computing, Singh et al. [14] introduced a unique load balancing technique called Vector Dot to tackle the hierarchical and multidimensional resource restrictions.

There are few resources available to help developers compare different resource scheduling techniques regarding the geographic distribution of both compute servers and user workloads, to assess the needs of large-scale cloud applications. In this study, we suggest using CloudSched for dynamic resource scheduling in a CDC to close the gap in tools for evaluating and modelling cloud environments and applications. CloudSched supports multiple scheduling algorithms, making it suitable for their application and comparison. Traditional scheduling algorithms consider only one aspect, such as the CPU, which can often result in hotspots or bottlenecks. CloudSched, however, treats multidimensional resources. In this chapter, the real-time restriction of both VMs and PMs—which is frequently disregarded in the literature—is discussed. This paper's primary contributions are:

1. putting forth a simulation system to simulate cloud computing environments and assess how well various resource scheduling policies and algorithms perform;

Creating and putting into practice a lightweight simulator that combines real-time, multidimensional resource data. Offering the following innovative characteristics is Cloud Sched:

1. Large-scale cloud computing ecosystems, including data centres, VMs, and PMs, are modelled and simulated.
2. Providing a framework for cloud IaaS to model various resource scheduling strategies and algorithms.
3. Support is provided for both graphical and text outputs.

The remaining sections of this chapter are structured as follows: Section 2 introduces the Cloud Sched architecture and its key components. The performance evaluations of various scheduling methods are covered in Section 3. The architecture and implementation of CloudSched are presented in Section 4. Section 5 compares several different scheduling strategies and presents the simulation findings. Finally, Section 6 offers conclusions.

II. RELATED WORK

The simplified layered architecture is shown in Figure 2:

1. Web portal. Users can choose resources and submit requests via a web interface at the top layer, where a few preset VM kinds are fundamentally available.
2. Core layer of scheduling. Once user requests are made, they move on to the next level of CloudSched scheduling, where the right data centres and PMs are chosen based on the user's requirements. In particular, CloudSched supports allocating VMs (consisting of CPU, memory, storage, bandwidth, etc.) to appropriate PMs while modelling and simulating CDCs. This layer is capable of managing massive CDCs comprising tens of thousands of PMs.

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Depending on the characteristics of the consumers, different scheduling algorithms might be used in various data centres.

3. Concentrating on the scheduling simulation on an IaaS layer, where relevant tools are still absent. Cloud-based resources at the lowest tier include PMs and VMs, each of which has a set amount of CPU, memory, storage, and bandwidth.

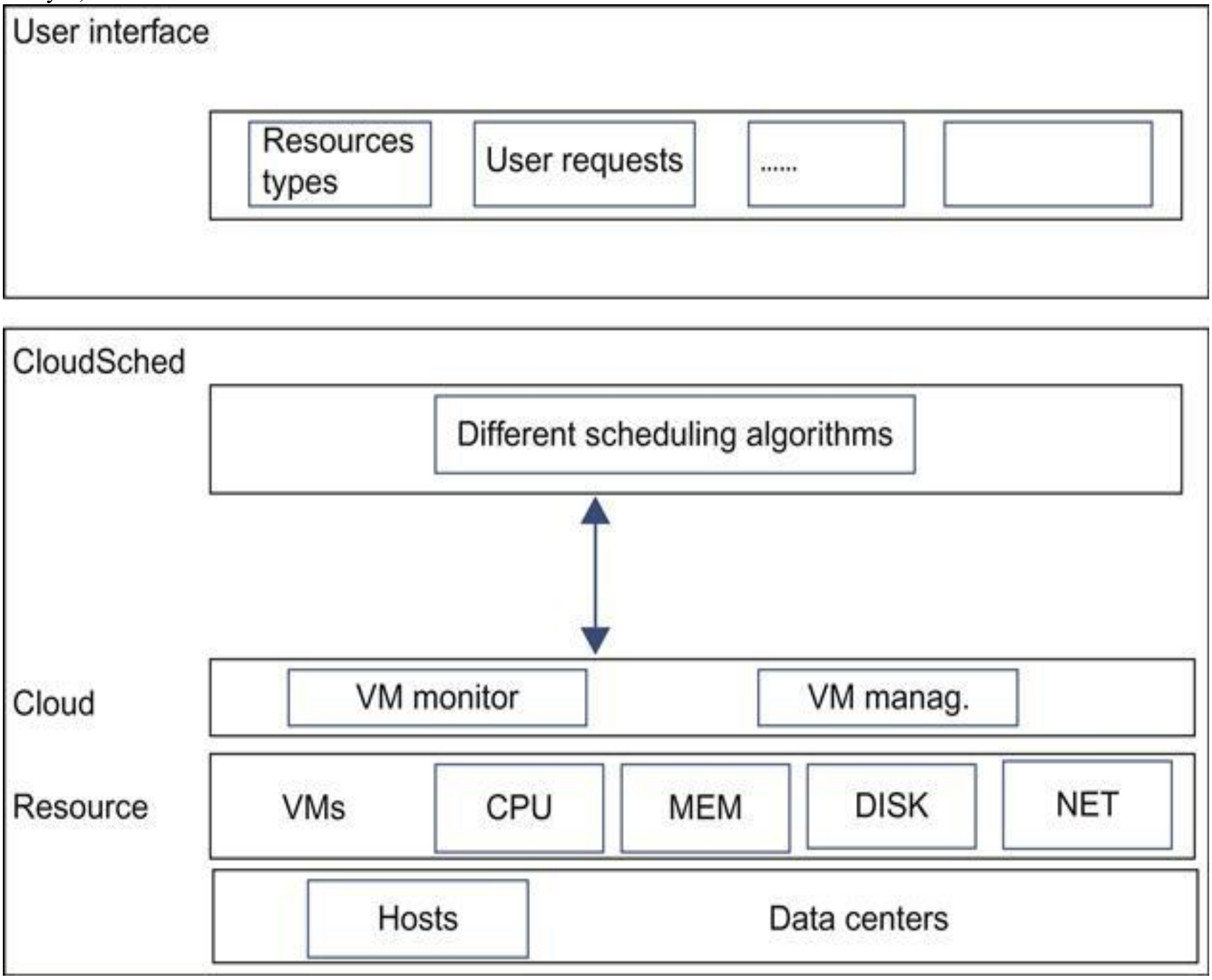

Figure 2. Simplified Layered Architecture of Cloud Sched.

The simulation system is extensive and complex because it is built upon several other tools, such as CloudSim and CloudAnalyst, which are based on existing simulation tools, including JavaSim and GridSim. These factors in mind, CloudSched focuses on resource scheduling methods and adopts a lightweight design. The main features of CloudSched are the following:

1. Focus on the IaaS layer. As opposed to current tools that concentrate on the application (task) level, such as CloudSim and CloudAnalyst, CloudSched concentrates on scheduling VMs at the IaaS layer, meaning that each request requires one or more VMs, whereas in CloudSim and CloudAnalyst, each request only uses a small portion of a VM's total capacity.
2. Providing a uniform view of all resources. CloudSched provides a unified view of all physical and virtual resources, simplifying system management and user selections, just like genuine Amazon EC2 applications do.
3. Lightweight design and scalability. CloudSched focuses on resource scheduling policies and algorithms in contrast to other simulation tools currently available, such as CloudSim and CloudAnalyst, which are built on GridSim (may cause issues). In a few minutes, Cloud Sched can mimic tens of thousands of queries.
4. High extensibility. Modular design is applied in CloudSched. For performance evaluation, several resource scheduling policies and algorithms can be implemented and compared with one another. An incredibly expansive distributed architecture can be created by modelling many CDCs.
5. Easy to use and repeatable. Simulator setup is quick and straightforward with CloudSched thanks to its user-friendly graphical user interface and outputs. Text files can be used as input sources and as output destinations. Modellers can repeat experiments by saving simulation inputs and outcomes with CloudSched. CloudSched guarantees that repeated simulations provide the same outcomes. Figures 3 and 4 illustrate several GUIs, respectively.

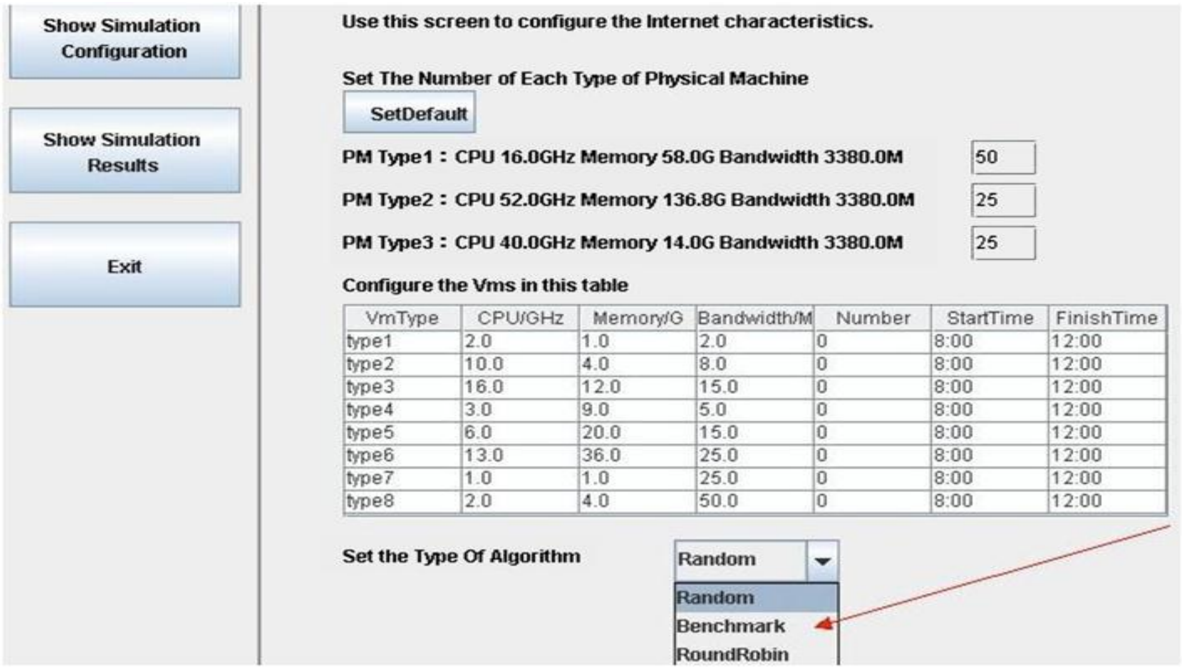

Figure 3. Main interface of Cloud Sched [1]

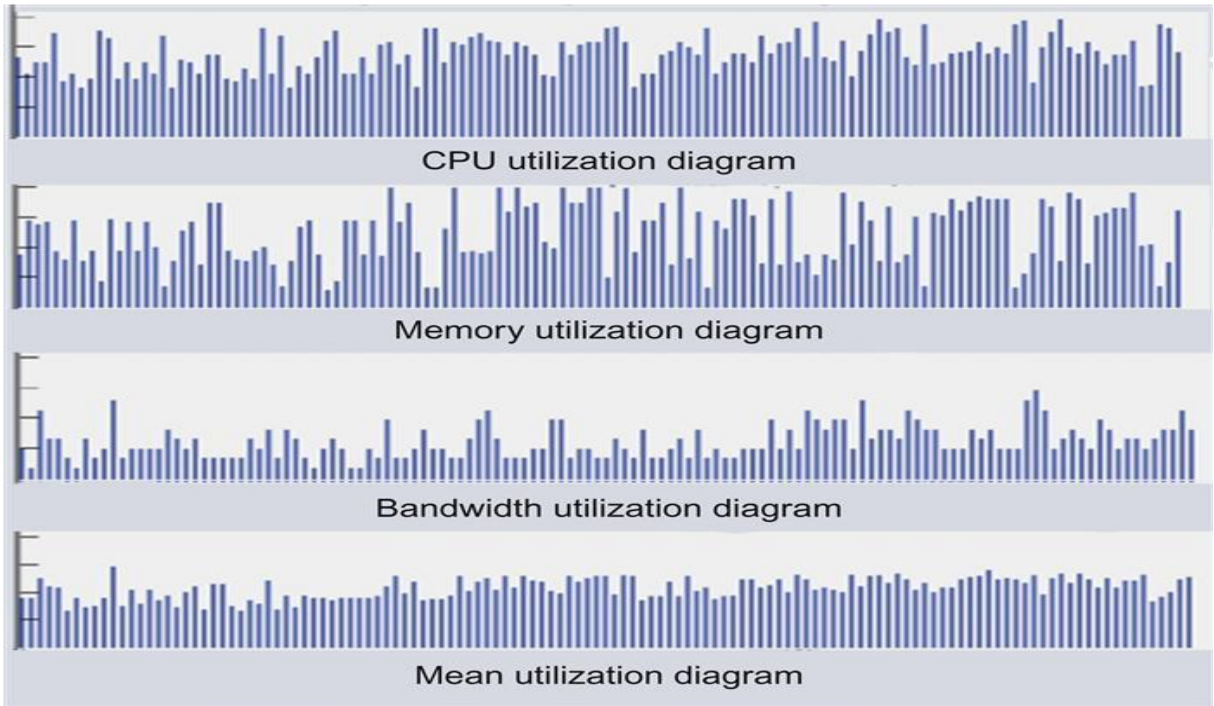

Figure 4. Main interface of Cloud Sched [2].

A. Modelling CDCs

A data centre component for managing VM requests models the main hardware architecture pertaining to clouds in the simulator. A data centre is primarily composed of several hosts, which are responsible for managing virtual machines (VMs) throughout their life cycles. A host is a cloud computing component that simulates a real computing node. It is given preconfigured processing power (measured in CPU units), memory, bandwidth, storage, and a scheduling policy for allocating processing cores to virtual machines. Similar representations are possible for VMs.

B. Modelling VM allocation

The flexibility in resource allocation offered by cloud computing is made possible by virtualisation technologies. A PM with two processing cores, for instance, can run two or more VMs simultaneously on each core. Virtual machines (VMs) can only be allocated if the total processing power consumed by all VMs on a host does not exceed the host's available processing power. We demonstrate that it is feasible to have a unified view of various VM types using the frequently used example of Amazon EC2. Eight different types of virtual machines are shown in Table 1.1 based on online data from Amazon EC2. Information on Amazon EC2's hardware setup is not available.

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

However, based on compute units, we can create three different sorts of PMs (or PM pools). For instance, a PM with 268.4 GB of memory, 16 cores, and 3.25 units of storage can be offered in a genuine CDC. This could lead to the formation of a unified perception of the various VM kinds.

However, based on compute units, we can create three different sorts of PMs (or PM pools). For instance, a PM with 268.4 GB of memory, 16 cores, and 3.25 units of storage can be offered in a genuine CDC. This could lead to the formation of a unified perception of the various VM kinds.

Table 1. Eight types of VMS in Amazon EC2

MEM	CPU (units)	BW (or Sto)	VM
1.7	1 (1 cores×1 unit)	160	1-1(1)
7.5	4 (2 cores×2 units)	850	1-2(2)
15.0	8 (4 cores×2 units)	1690	1-3(3)
17.1	6.5 (2 cores×3.25 units)	420	2-1(4)
34.2	13 (4 cores×3.25 units)	850	2-2(5)
68.4	26 (8 cores×3.25 units)	1690	2-3(6)
1.7	5 (2 cores×2.5 units)	350	3-1(7)
7.0	20 (8 cores×2.5 units)	1690	3-2(8)

Table 2. Three types of PMs are suggested

PM	CPU (units)	MEM	BW (or Sto)
1	16 (4 cores×4 units)	160	1-1(1)
2	52 (16 cores×3.25 units)	850	1-2(2)
3	40 (16 cores×2.5 units)	1690	1-3(3)

The current scheduling techniques used by CloudSched include dynamic load balancing, optimal utilization, and energy-efficient scheduling. It is also possible to use other algorithms, like reliability- and cost-oriented ones.

C. Modelling customer requirements

By randomly generating various VM types and assigning VMs based on suitable scheduling algorithms across multiple data centres, CloudSched models customer requirements. Random processes can be used to produce the arrival process, service time distribution, and necessary capacity distribution of requests. It is possible to regulate the rate at which consumer inquiries arrive. It is also possible to distribute the various VM requirements across multiple servers. An interval vector called vmID can be used to describe a real-time VM request. For instance, the expression vm1(1, 0, 6, 0.25) indicates that the request ID is 1, the VM is of type 1 (equivalent to integer 1), the start time is 0, the finish time is 6 (here, 6 can represent that the sixth slot ended at time 6), and the capacity that a VM occupies from a specific PM is 0.25. Similar representations can be used for other requests. Figure 5 illustrates the life cycles of VM allocation using two PMs in a slotted time window. PM1 hosts VMs 4, 5, and 6, while PM2 hosts VMs 1, 2, and 3.

III. MATERIALS AND METHODOLOGY

CloudSched treats multidimensional resources, such as CPU, memory, and network bandwidth, as integrated for both PMs and VMs, unlike typical scheduling algorithms that only consider one component at a time, which can often result in hotspots or bottlenecks. There aren't enough relevant metrics available for scheduling algorithms that consider multiple dimensions. There are several metrics for various scheduling goals. The measurements for load balancing, energy efficiency, and utilisation are discussed in the following sections. It is simple to add new metrics for additional goals.

IV. THEORY/CALCULATION

A. Metrics for multidimensional load balancing

Following an examination of various current metrics, we create an integrated measurement that accounts for both the average imbalance level across all servers and the overall imbalance level of the CDC. A few VM migration approaches were introduced by Wood et al. [13]. The following is how one integrated load balance metric is used:

$$V = \frac{1}{(1-\mathrm{CPU}_u)(1-\mathrm{MEN}_u)(1-\mathrm{NET}_u)} \quad (1)$$

where CPUu, MENu, and NETu represent the average CPU, memory, and network bandwidth usage during each observed period, respectively. The higher the combined usage, the larger the value of V. Thus, this measurement can serve as the foundation for migration algorithms. By transforming three-dimensional (3D) resource information into one-dimensional (1D) values, this technique aims to reduce the integrated use of resources. Information in many dimensions could be lost during this transfer. Zheng et al. [16] proposed another integrated load balancing metric as follows:

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

$$B = \frac{aN1_iC_i}{N1_mC_m} + \frac{bN2_iM_i}{N2_mM_m} + \frac{cN3_iD_i}{N3_mD_m} + \frac{\mathrm{dNet}_i}{\mathrm{Net}_m} \quad (2)$$

First, the suggested physical server m is chosen. Then server m is contrasted with other physical servers i. The CPU capability is N1i, the memory capability is N2i, and the hard drive is N3i. Here, Ci and Mi represent the average CPU and memory usage, respectively. Di represents the hard disk transfer rate, and Neti represents the network throughput. The weighting factors for the CPU, RAM, hard drive, and network bandwidth are indicated above by the letters a, b, c, and d, respectively. The primary objective of this algorithm is to allocate virtual machines to physical servers with the lowest value of B. This method also transforms 3D resource data into a 1D value.

To take into account integrating variables of load balance for flow channels in data centres, Singh et al. [14] devised a novel Vector Dot algorithm. The average CPU, memory, and network bandwidth usage of a server are shown by the node fraction vectors CPUU, memU, and netU, respectively. The terms CPUCap, memCap, and netCap, respectively, refer to the overall CPU, memory, and network bandwidth capabilities of a server. The node utilisation threshold vector is represented by CPUT, memT, netT, and ioT, where CPUT, memT, netT, and ioT, respectively, stand for the CPU, memory, and network bandwidth usage thresholds. The concept of an imbalance score is used to assess the level of overload in a node and the system. A node's imbalance score is determined by:

$$\mathrm{IBscore}(f, T) = \begin{cases} 0, & \text{if} \quad f < T \\ e^{(f-T)/T}, & \text{otherwise} \end{cases} \quad (3)$$

The system's overall imbalance score is calculated by adding the scores of all the nodes. This nonlinear measurement has the advantage of being able to distinguish between two pairs of nodes that are both at 2T and 3T. A useful metric for comparing average usage to its threshold is the imbalance score. An integrated measurement for the total imbalance level of a CDC, as well as the average imbalance level of each server, has been designed for load balancing techniques after considering the benefits and drawbacks of existing metrics for resource scheduling. There is also room for developing more measures for other scheduling approaches. The following criteria are taken into account:

1. A single server's typical CPU use is i. The average CPU usage throughout an observational period is what is meant by this. The average of six recorded values for server i, for instance, is calculated if the observing period is 1 minute and the CPU utilisation is recorded every 10 seconds.
2. Average CPU use across the whole CDC. Suppose that there are CPUs on server i in total.

The system's overall imbalance score is calculated by adding the scores of all the nodes. This nonlinear measurement has the advantage of being able to distinguish between two pairs of nodes that are both at 2T and 3T. A useful metric for comparing average usage to its threshold is the imbalance score. An integrated measurement for the total imbalance level of a CDC, as well as the average imbalance level of each server, has been designed for load balancing techniques after considering the benefits and drawbacks of existing metrics for resource scheduling. There is also room for developing more measures for other scheduling approaches. The following criteria are taken into account:

1. A single server's typical CPU use is i. The average CPU usage throughout an observational period is what is meant by this. The average of six recorded values for server i, for instance, is calculated if the observing period is 1 minute and the CPU utilisation is recorded every 10 seconds.
2. Average CPU use across the whole CDC. Suppose that there are CPUs on server i in total.

$$\mathrm{CPU}_u^A = \frac{\sum_i^N \mathrm{CPU}_i^U \mathrm{CPU}_i^n}{\sum_i^N \mathrm{CPU}_i^n} \quad (4)$$

where N is the total number of physical servers in a CDC. Similarly, the average utilisation of memory and network bandwidth of server i, as well as all memories and all network bandwidth in a CDC, can be defined as follows.

3. ILBi, or integrated load imbalance value, for server I. In statistics, variance is frequently used to quantify the dispersion of a collection of numbers from one another. An integrated load imbalance value (ILBi) of server I is defined using variance:

$$\frac{(\mathrm{Avg}_i - \mathrm{CPU}_u^A)^2 + (\mathrm{Avg}_i - \mathrm{MEM}_u^A)^2 + (\mathrm{Avg}_i - \mathrm{NET}_u^A)^2}{3} \quad (5)$$

Where

$$\mathrm{Avg}_i = \frac{(\mathrm{CPU}_i^U + \mathrm{MEM}_i^U + \mathrm{NET}_i^U)}{3} \quad (6)$$

(ILBi) is a term used to describe the degree of load imbalance when comparing a single server's CPU, memory, and network bandwidth usage.

4. The imbalance value of all CPUs, memories, and network bandwidth. Using variance, the imbalance value of all CPUs in a data centre is defined as

$$\mathrm{IBL}_{\mathrm{cpu}} = \sum_i^N (\mathrm{CPU}_i^U - \mathrm{CPU}_u^A)^2 \quad (7)$$

Memory and network bandwidth imbalance values can also be computed. The total imbalance values across all servers in a CDC are then calculated as follows:

$$\mathrm{IBL}_{\mathrm{tot}} = \sum_i^N \mathrm{ILB}_i \quad (8)$$

5. Average imbalance value of physical server i. A physical server's average imbalance value is defined as:

$$\mathrm{IBL}_{\mathrm{avg}}^{\mathrm{PM}} = \frac{\mathrm{IBL}_{\mathrm{tot}}}{N} \quad (9)$$

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Where N represents the total number of servers, this statistic is used to gauge the level of imbalance on all physical servers, as its name implies.

6. Average imbalance value of a CDC. The average imbalance value of a CDC is defined as

$$IBL_{avg}^{CDC} = \frac{IBL_{cpu} + IBL_{mem} + IBL_{net}}{N} \tag{10}$$

7. Average times for running. It is possible to compare the average running times of the same number of jobs for various scheduling algorithms.
8. Makespan. The maximum load (or average utilization) for each PM is what is meant by this.
9. Utilization efficiency. This is determined in this instance by dividing the minimum load on any PM by the highest load on any PM.

B. Metrics for energy efficiency

Power consumption model

1. The server's estimated power usage model. Data centres' cooling, disk storage, networks, and computation systems use the most electricity. The authors proposed a power consumption model for blade servers in Reference [17], where P is defined as

$$14.5 + 0.2U_{cpu} + (4.5E-8)U_{men} + 0.003U_{disk} + (3.1E-8)U_{net} \tag{11}$$

where UCPU, Umem, Udisk, and Unet represent, respectively, the utilization of the CPU, memory, hard drive, and network interface. Other elements, such as RAM, hard drives, and network interfaces, have a minimal effect on overall power consumption. The authors of Ref. [3] discovered that CPU usage is frequently proportional to the total system load and proposed the following power model:

$$P(U) = kP_{max} + (1-k)P_{max}U \tag{12}$$

Where U is the CPU usage, k is the percentage of power consumed by the idle server (studies show that, on average, it is roughly 0.7). Pmax is the maximum power consumed when the server is fully employed. In contrast to other resources, such as memory, disk storage, and network devices, the CPU consumes the majority of the energy in this paper. Because workloads fluctuate in the real world, CPU utilization may change over time. The CPU utilization is therefore a function of time and is denoted by the symbol u. As a result, the total energy used by a PM (Ei) can be calculated as an integral of the power consumption function over time using the formula:

$$E_i = \int_{t_0}^{t_1} P(u(t))dt \tag{13}$$

If $u(t)$ is constant over time (e.g., average utilization is adopted, $u(t)=u$), then $E_i=P(u)(t_1-t_0)$.

2. The total energy consumption of a CDC is computed as

$$E_{cdc} = \sum_{i=1}^{n} E_i \tag{14}$$

It is the total amount of energy used by all PMs. It should be noted that the energy usage of all VMs running on PMs is included.

The CPU utilization is therefore a function of time and is denoted by the symbol u.

3. The overall quantity of PMs used. The total number of PMs utilized for the specified set of VM requests is shown below. It is crucial for energy effectiveness.
4. The entire amount of time that each PM was powered on. The total power-on time is the most important aspect according to the energy consumption equation of each PM.

C. Metric for maximizing resource utilization

1. Average use of the resources. You may compute the average usage of the CPU, memory, hard drive, and network bandwidth. You can also utilize the combined usage of all these resources.
2. The overall quantity of PMs used. It is directly tied to a CDC's average and overall utilization.

V. RESEARCH DESIGN AND IMPLEMENTATION OF CLOUDSCHED

We give information about the conception and execution of CloudSched in this part. The implementation is a Java discrete simulator. The following is a brief description of CloudSched's main components.

A. IaaS resources are considered

IaaS resources considered in this chapter include:

1. PMs: The actual computers that make up data centres. Each PM can manage a number of virtual machines, and each PM can have different combinations of CPU, memory, hard drives, network cards, and other associated parts.
2. Physical clusters: These are composed of the required network, storage, and the number of PMs.
3. VM: an on-the-PM virtual computing environment that makes use of virtualization software. There are several virtual CPUs, as well as memory, storage, network cards, and related hardware elements.
4. Virtual cluster: includes a number of virtual machines (VMs) and the required network and storage infrastructure.

B. Scheduling process in CDC

A typical architecture of CDCs and key resource scheduling procedures is shown in Figure 6:

1. User requests: Through the internet, the user makes the request (for example, by logging into the online portal of the cloud service provider).
2. Scheduling management: Based on the user's identity (e.g., location) and the operational details of the request, the Scheduler Centre makes choices. The appropriate data centre receives the request, which is then sent to the Scheduler Centre through the data centre management program. Based on scheduling methods used in CDCs, the Scheduler Centre allocates the request.
3. Feedback: The user is given access to the resources through the scheduling mechanism.
4. Execute scheduling: The following stage receives the scheduling outcomes (such as deploying actions).

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

5. Updating and optimisation: According to the optimising objective functions, the scheduler optimises resources across several data centres by updating resource information.

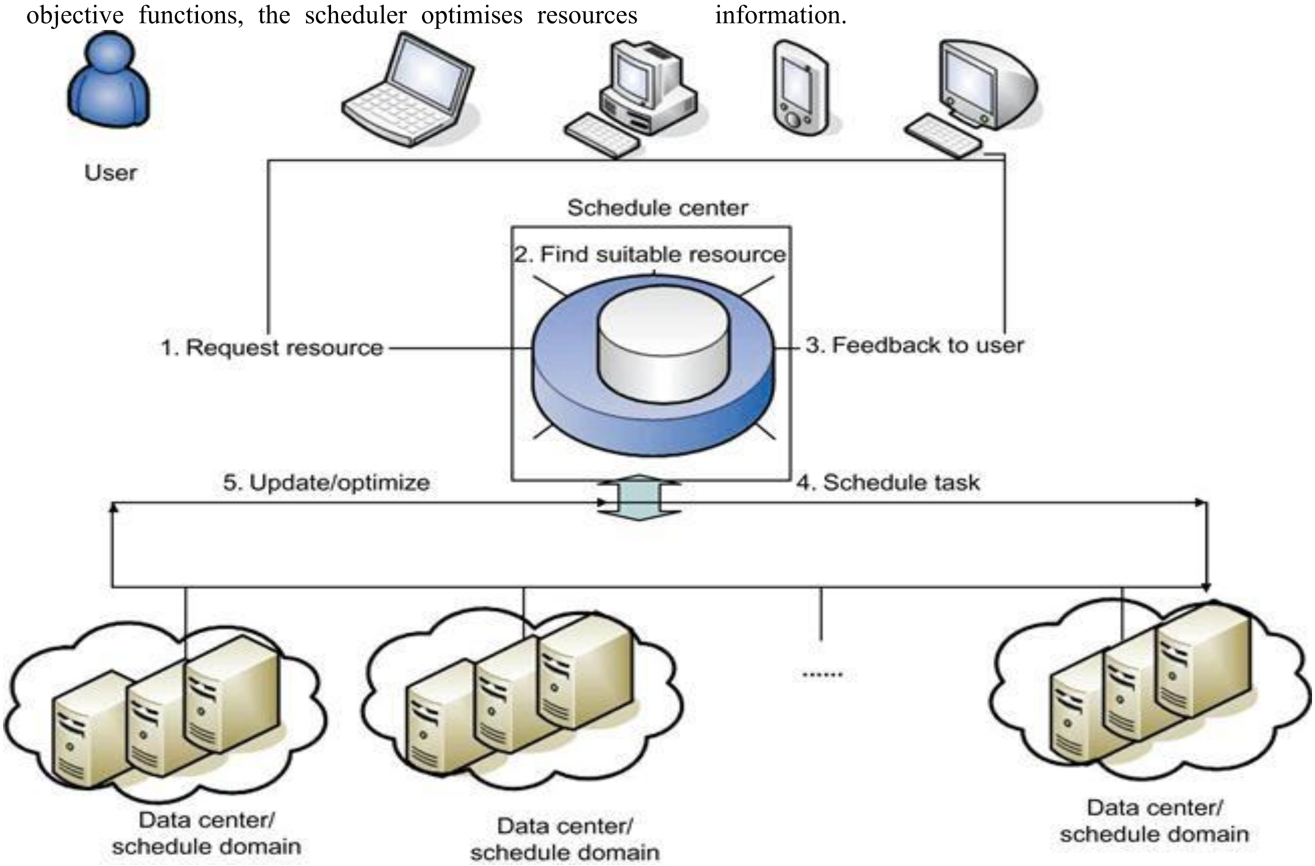

Fig. 5. Referred architecture of CDCs.

The primary resources in CDCs are shown in general and detailed UML diagrams, respectively, in Figures 7 and 8. Figure 7 illustrates the primary resources and their connections within CDCs, while Figure 8 details the characteristics of each primary resource. The core methods of the ScheduleDomain class manage jobs in each queue by invoking other tasks. Task requests are generated by the classes CreateRandVM and VmTaskInfo. Requests from VMs are allocated using the Class Allocate and Sort method.

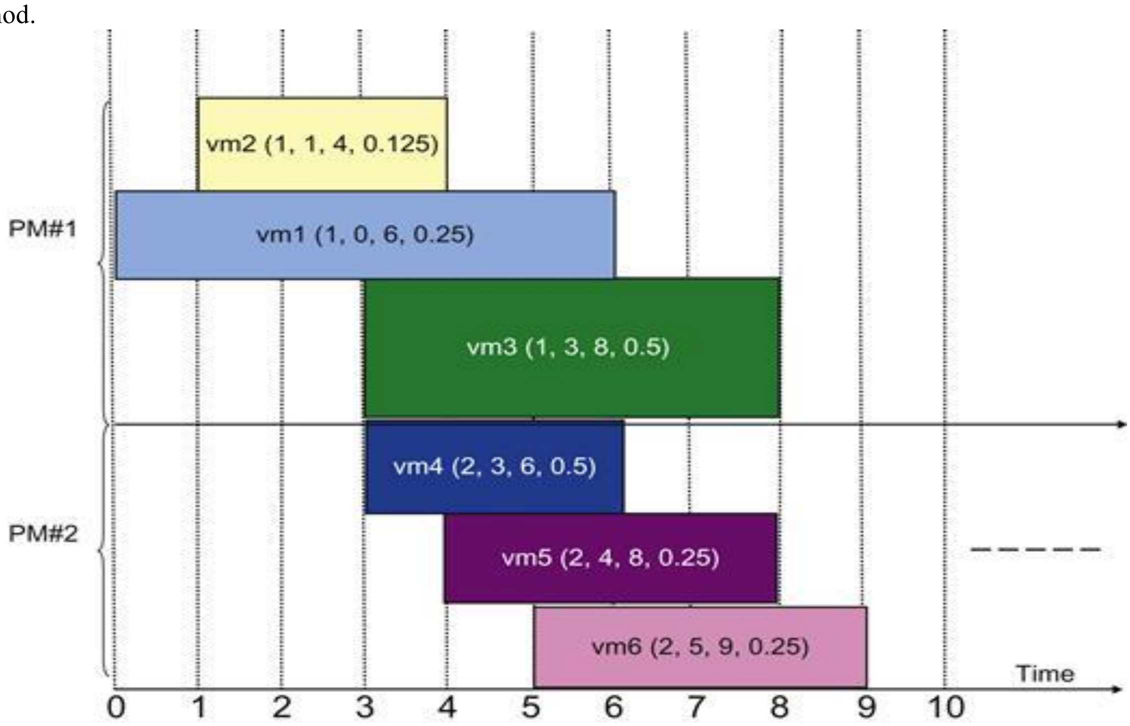

Figure 6. Example of user request

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

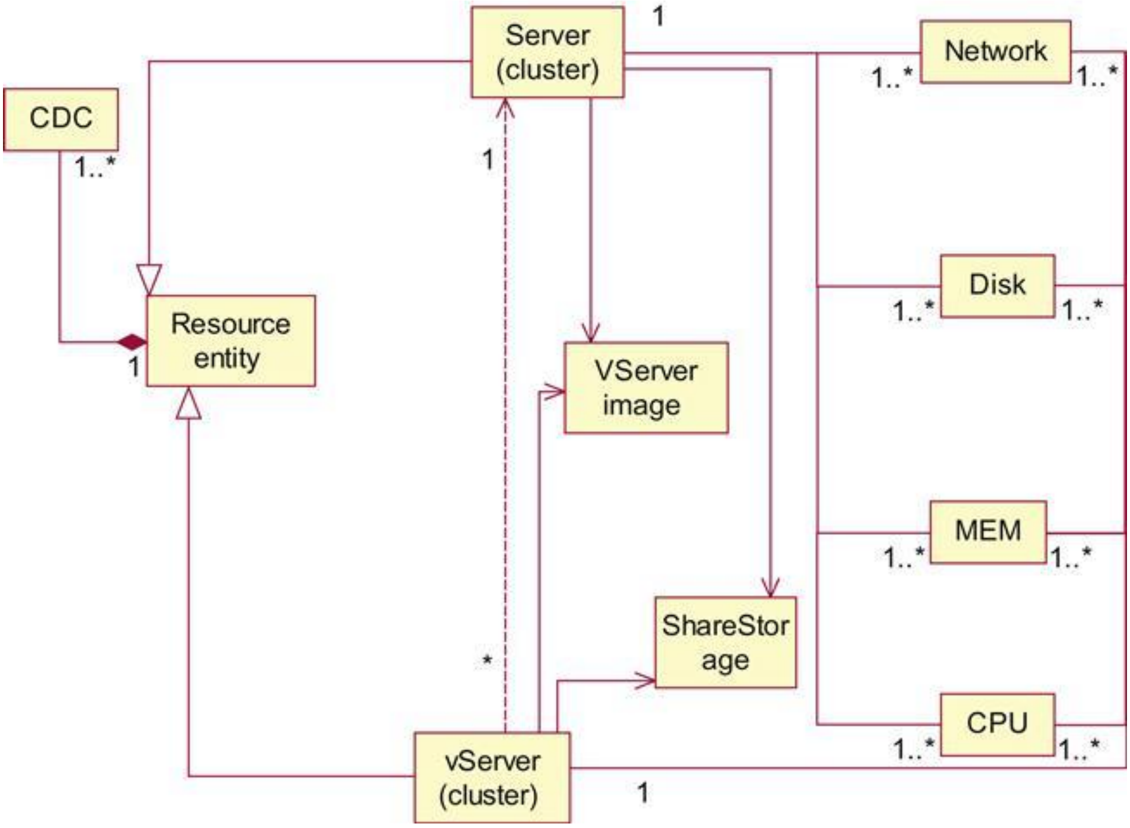

Figure 7. UML diagram of primary resources in CDCs.

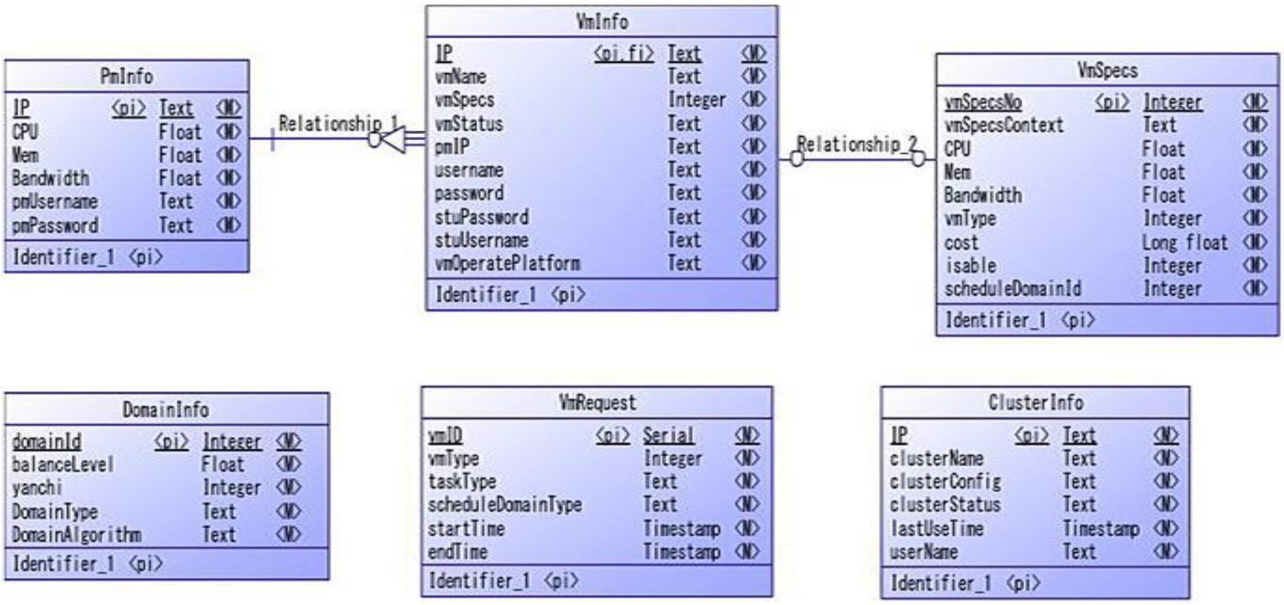

Figure 8. Detailed UML diagram of primary resources in CDCs.

C. Scheduling algorithms: taking the LIF algorithm as an example

The pseudocode for the Least Imbalance Level First (LIF) algorithm, which supports a CDC's dynamic load balance, is shown in Figure 9. The method takes as inputs the state of the currently active tasks, PMs, and the current VM request r. The placement scheme for request r is what is produced via dynamic scheduling. Essentially, when a new VM request is submitted, the system compares several imbalance numbers to determine which PM has the lowest total imbalance value for the data centre. The algorithm finds the PM with the lowest integrated load. The result will be the lowest overall imbalance number across all servers in a CDC. The placement scheme for request r is what is produced via dynamic scheduling. Essentially, when a new VM request is submitted, the system compares several imbalance numbers to determine which PM has the lowest total imbalance value for the data centre.

```
Algorithm : Lowest-Average-Value-First(R)
Input: placement request r = (id,t_s,t_e.k);
         status of current active tasks and PMs
Output: placement scheme for r and IBL_tot.
1) initialization: LowestAvg = large number;
2) For i=1:N Do
3)    If  request r can be placed on PM (i)
4)    Then
5)      compute  avg(i)  utilization value of PM(i) it using    equations  (4)-(6);
6)        If avg(i)<LowestAvg
7)          Then
8)            LowestAvg=avg(i);
9)            allocatedPMID=i;
10)             Else
11)           EndIf
12)       Else  //find next PM
13)   Endfor
14)   IF LowestAvg== large number L  // cannot allocate
15)       Then put r into waiting queue or reject
16)        Else place r on PM with allocatedPMID and compute IBL_tot
```

Figure 9. LIF algorithm.

The LIF algorithm's primary class diagram and sequence diagram are shown in Figures 10 and 11, respectively.

The core methods of the Schedule Domain class manage jobs in each queue by invoking other courses.

Task requests are generated by the classes CreateRandVM and VmTaskInfo. Requests from VMs are allocated using the Class Allocate and Sort method.

VMs can be moved using Class Migrate and Allocate-Alg. Record, Print PM, and Balance Level handle printing and output tasks. Physical servers and virtual machines are operated by the server, PM, and VM.

LIF algorithm: Always chooses PMs with the lowest integrated imbalance value (as stated in Eq. (5)) and available resources to assign VMs based on needs characteristics (e.g., CPU intensive, high memory, high bandwidth requirements, etc.).

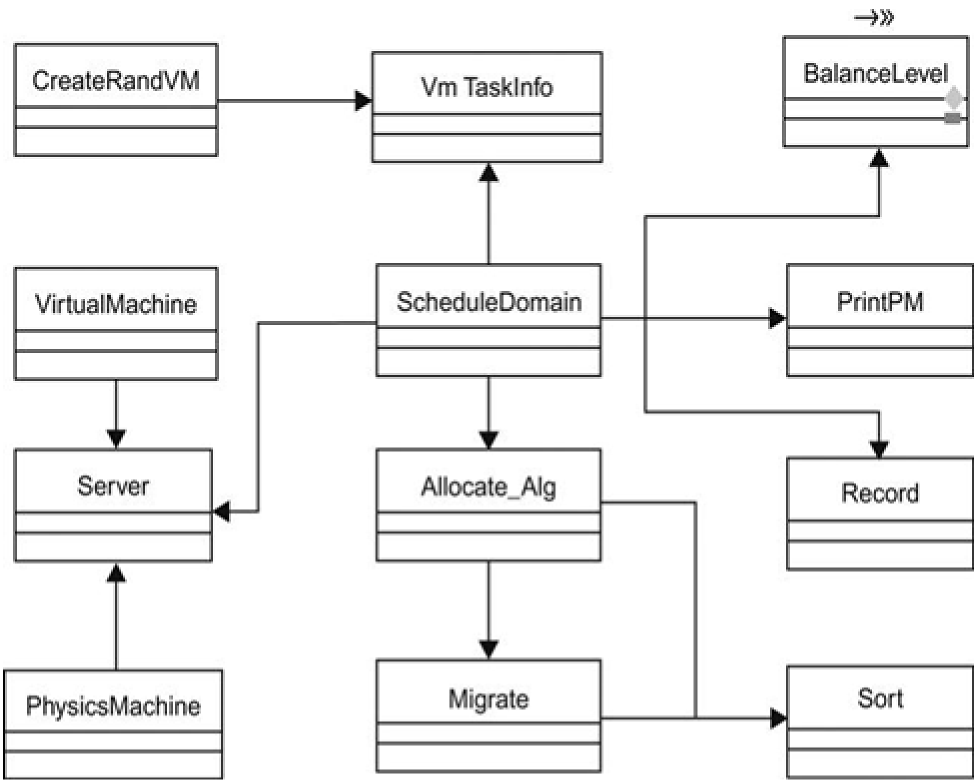

Figure 10. Main class diagram.

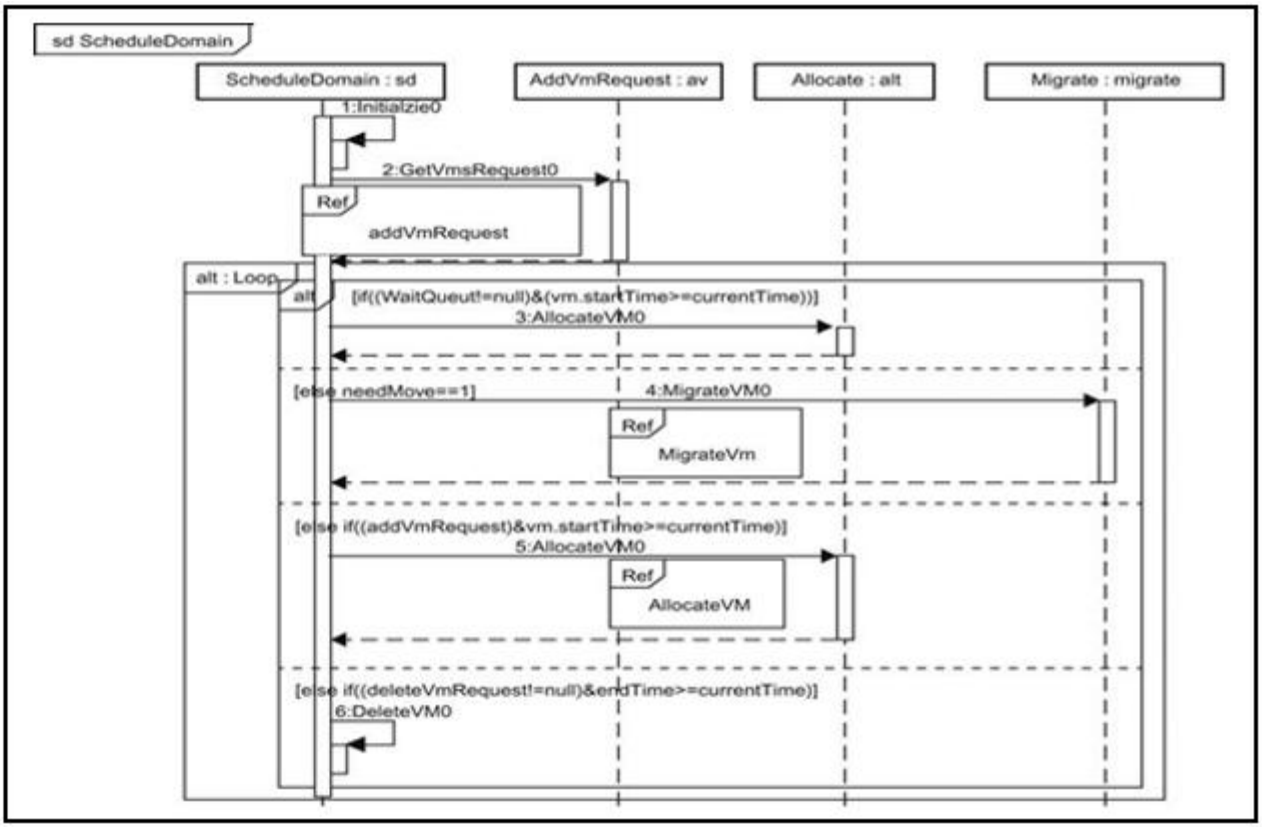

Figure 11. Sequence diagram.

A sequence diagram shows the following sequence of the algorithm:

1. Initialize the system;
2. Obtain task requests;
3. Allocate VM requests in the waiting queue;
4. Operate migrating queues;
5. Operate requesting queues;
6. Operate deleting queues;

One of the interfaces for installing CDCs in CloudSched is shown in Figure 2. The manager first selects a data centre using various IDs, after which the quantity and type of PMs are configured. Data centres may be added or removed by the manager. One of the user request configuration interfaces is shown in Figure 13. It is possible to set up probability distributions for the various VM types, the overall number of simulated VMs, and preferred data centres. Figure 10 shows the design diagram of the primary classes.

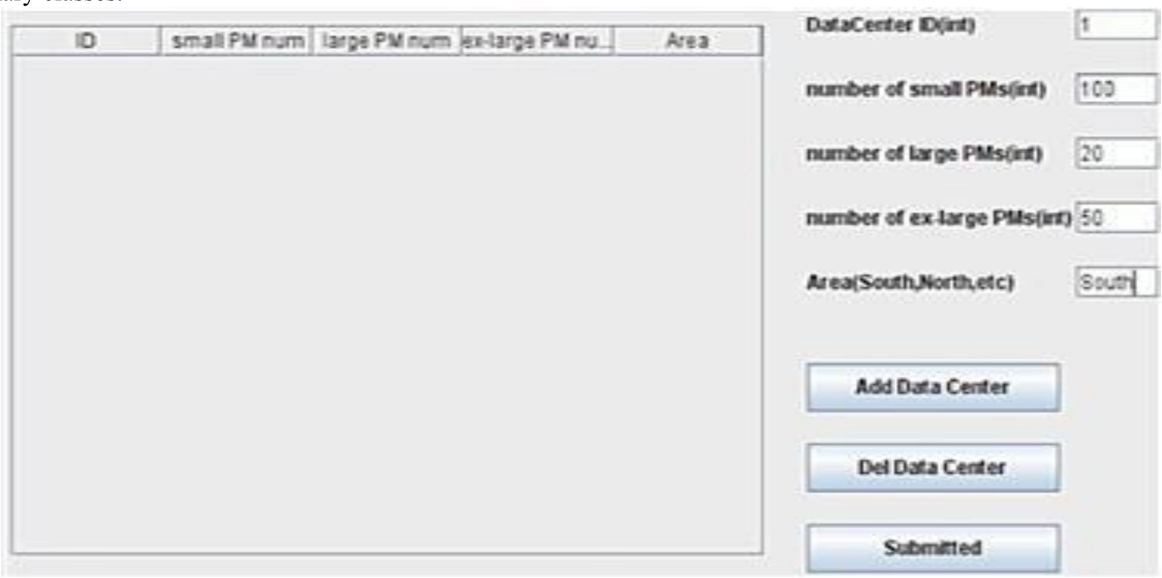

Figure 12. One interface for configuring CDCs.

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

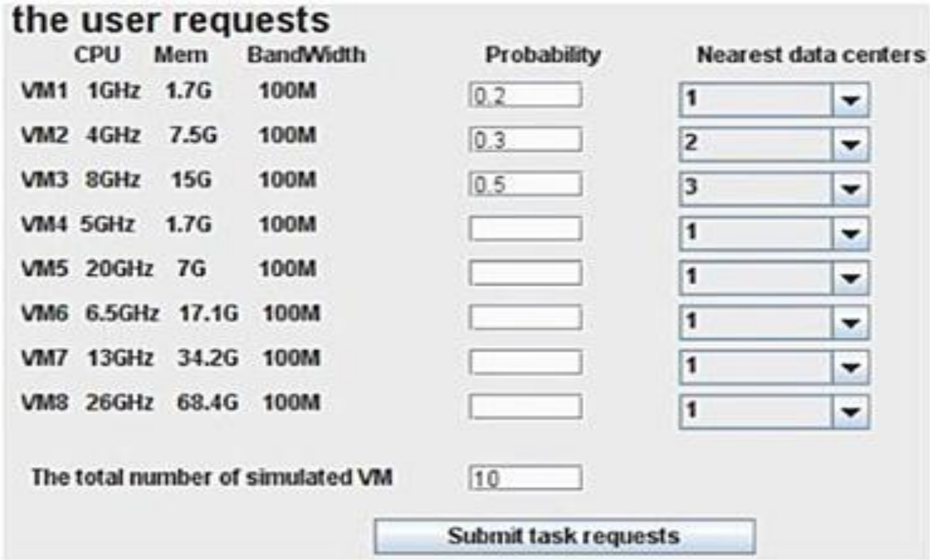

Figure 13. One interface for configuring user requests.

VI. RESULT AND DISCUSSION

For the simulation, we utilize a standard Pentium PC with a 2 GHz CPU and 2 GB of RAM.

A. Random configuration of VMs and PMs

In this section, we compare simulation results from four distinct load-balancing scheduling strategies. For your convenience, a short name for each algorithm is provided as follows:

1. ZHCJ algorithm: As described in Ref. [16], the method allocates VMs to PMs with the available resources and the lowest V value (as stated in Eq.
2. To allocate VMs, the ZHJZ method chooses a reference PM [16], calculates the value, and then selects PMs with the lowest B value (as given in Eq. (2)) and available resources.
3. LIF algorithm: Always chooses PMs with the lowest integrated imbalance value (as stated in Eq. (5)) and available resources to assign VMs based on needs characteristics (e.g., CPU intensive, high memory, high bandwidth requirements, etc.).
4. Requests (VMs) are assigned at random to PMs with available resources using the Rand algorithm.

Round-Robin algorithm: One of the simplest scheduling algorithms, it assigns tasks to each physical server in equal portions and circular order, handling all tasks without priority (also known as cyclic executive).

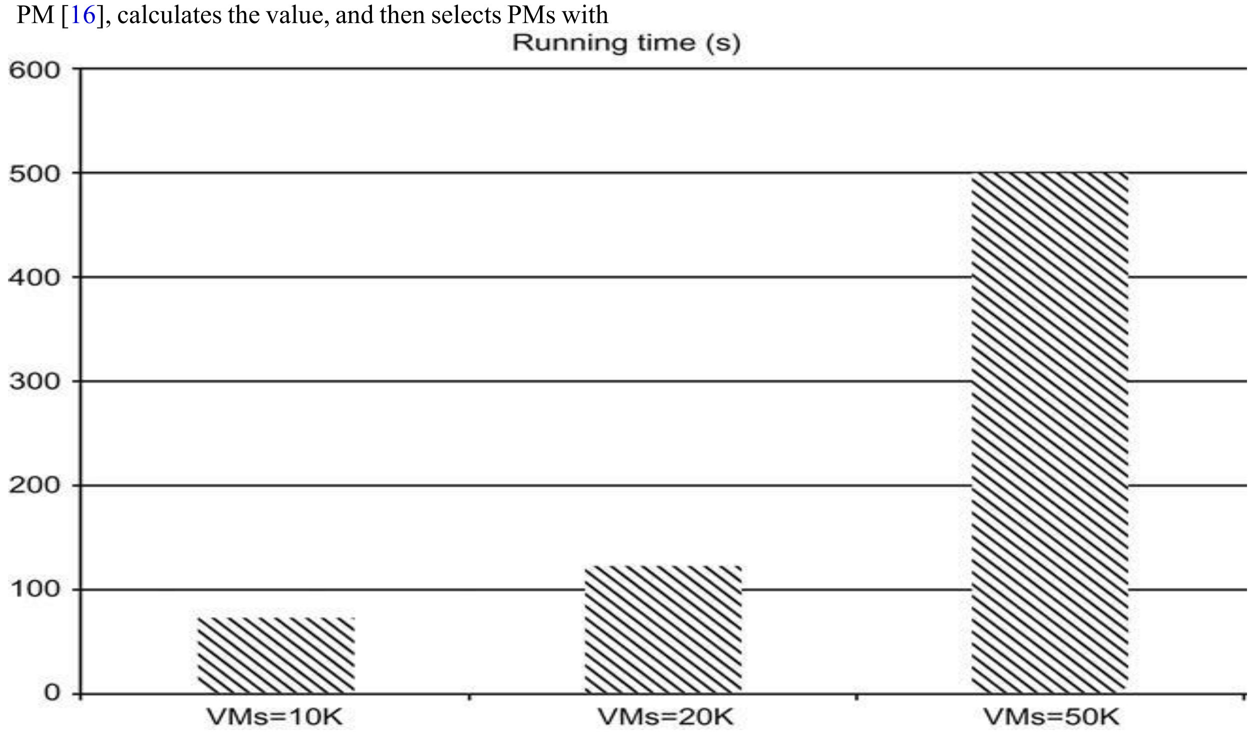

Figure 14. Running time of Cloud Sched.

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Three different types of heterogeneous PMs are taken into account for the simulation, and each PM pool contains a certain number of PMs. Both CPU and RAM are configured with a large size, which may be set dynamically, for the simulation of a lot of VM requests:

PM type 1: CPU=6 GHz, memory=8 G, and bandwidth=1000 M

PM type 2: CPU=12 GHz, memory=16 G, and bandwidth=1000 M

PM type 3 CPU=18 GHz, memory=32 GB, and bandwidth=1000 Mbps.

Eight different types of virtual machines (VMs) with equal probabilities of requests are generated at random and are similar to eight Amazon EC2 instances with high CPU, high RAM, and standard specifications (but not exactly the same) as follows (may be dynamically configured):

Type 1: CPU=1.0 GHz, memory=1.7 G, bandwidth=100 M

Type 2: CPU=4.0 GHz, memory=7.5 G, bandwidth=100 M

Type 3: CPU=8.0 GHz, memory=15.0 G, bandwidth=100 M

Type 4: CPU=5.0 GHz, memory=1.7 G, bandwidth=100

Type 5: CPU=20.0 GHz, memory=7.0 G, bandwidth=100 M

Type 6: CPU=6.5 GHz, memory=17.1 G, bandwidth=100 M

Type 7: CPU=13.0 GHz, memory=34.2 G, bandwidth=100 M

Type 8: CPU=26.0 GHz, memory=68.4 G, and bandwidth=100 M.

All simulations are conducted on a Pentium PC with a 2 GHz CPU and 2 GB of RAM, utilising a range of PM counts from 100 to 600 and VM request counts from 1000 to 6000. The eight different types of virtual machines (VMs) listed earlier are all considered equally frequently while generating the input data for user requests. Of course, various (random) probabilities for different VM types can be produced. For steady-state analysis, the transitory period is dropped in favour of a warm-up period (the first 2000 queries). Figure 15 shows the average imbalance level, defined in Eq. (10), of a CDC. It can be seen that the LIF algorithm has the lowest average imbalance level when the total number of VMs and PMs is varied.

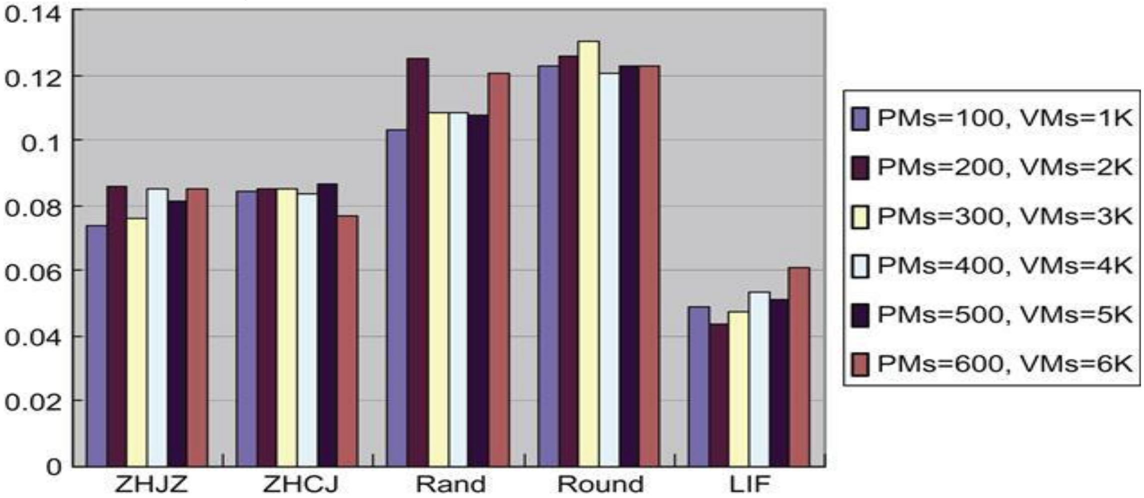

Figure 15. Average imbalance values of a CDC.

Figure 16 displays the overall physical server's average imbalance level as stated by Eq. When the combined number of VMs and PMs is changed, the LIF algorithm once more has the lowest average imbalance level for all PMs.

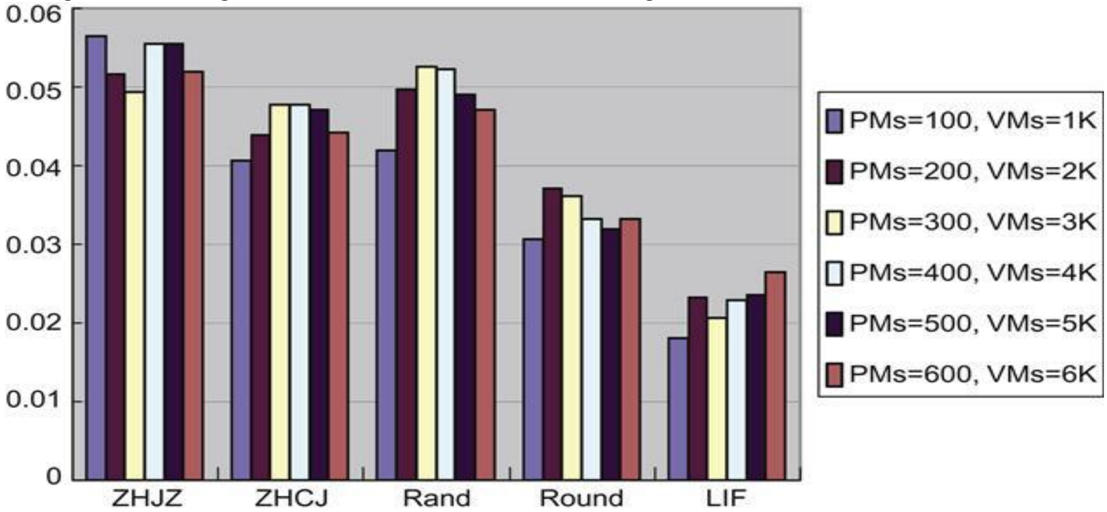

Figure 16. Average imbalance values of each physical server.

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

When the total number of physical servers is fixed but the number of VMs is variable, Figure 17 depicts the average imbalance level of a CDC, as specified in Eq. (10).

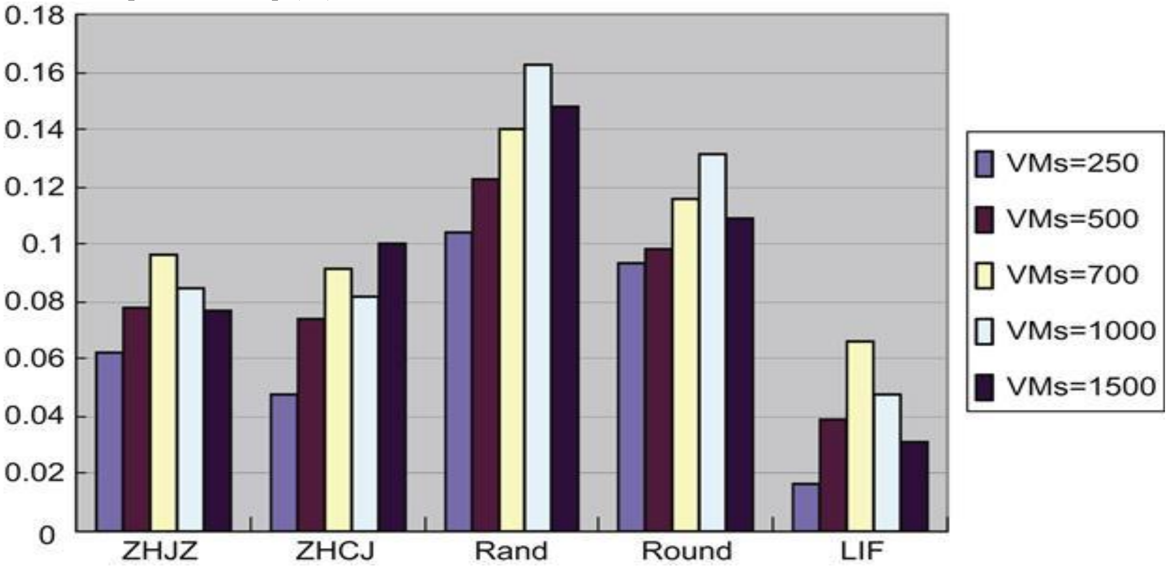

Figure 17. Average imbalance values of a CDC when PMs=100.

When the overall number of physical servers is fixed but the number of VMs is variable, Figure 18 displays the average imbalance level of the entire physical server, as described in Eq. (5). Similar outcomes are observed after a lengthy simulation.

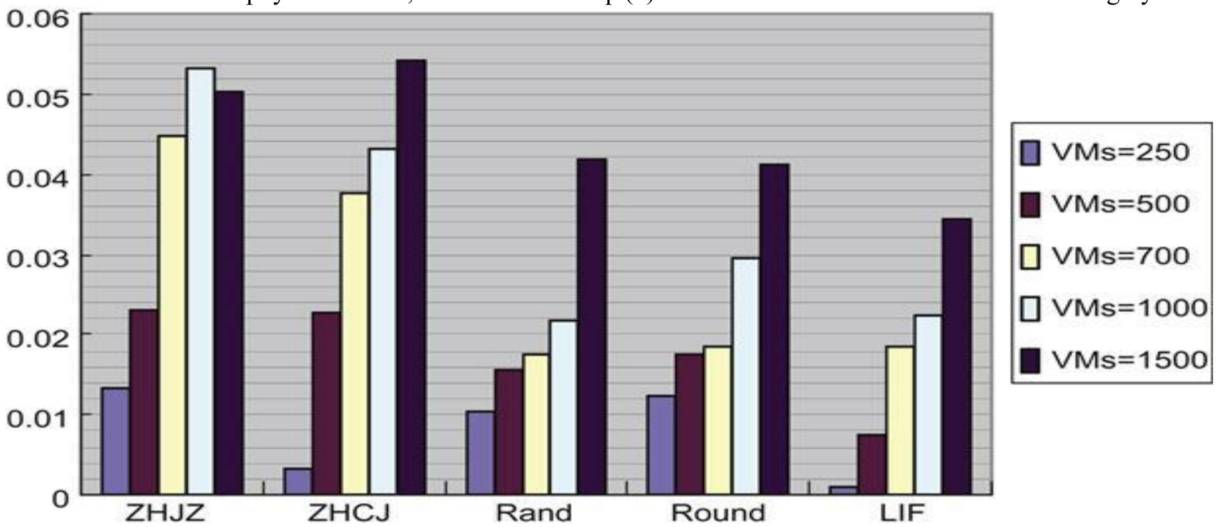

Figure 18. Average imbalance values of each physical server when PMs = 100.

B. Divisible size configuration of PMs and VMs

Section 2.2 explains how VMs and PMs are configured. We display the average CPU, memory, and bandwidth utilisation, as well as the average of these three, in Figures 19–21. We also demonstrate the overall data centres' imbalance value (IBL, as in Eq. (10), using five distinct algorithms: Round-Robin, ZHJZ, ZHCJ, and LIF. As can be shown, LIF consistently has the most significant average CPU, memory, and bandwidth consumption but the lowest imbalance value (when the total number of VMs and PMs varies). These findings demonstrate that metrics derived from divisible situations are substantially more accurate than those from random configuration cases. As a result, cloud service providers like Amazon can utilise these settings to meet better customer demands for load balancing, energy efficiency, and other performance-related needs.

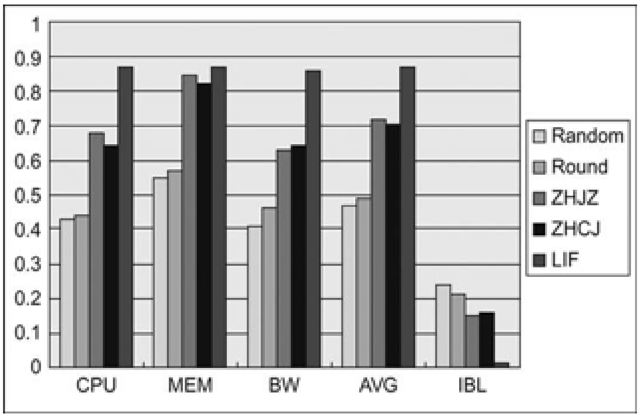

Figure 19. Utilization and imbalance value of the entire data center when PMs=100 and VMs=1000.

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

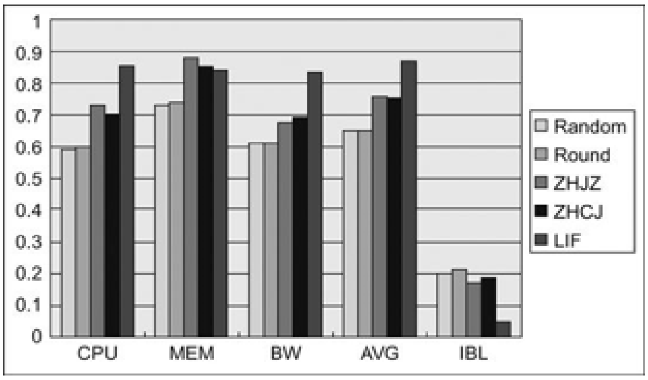

Figure 20. Utilization and imbalance value of the entire data center when PMs=200 and VMs=4000.

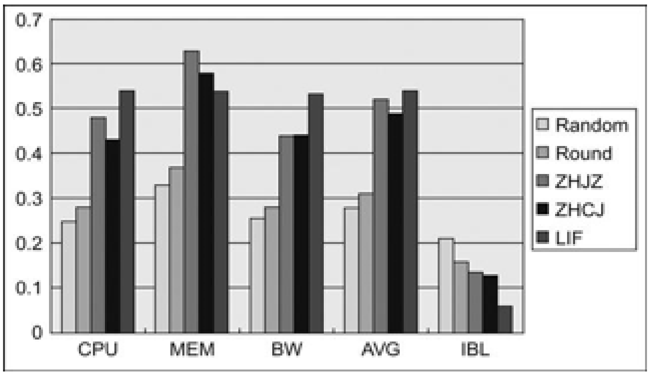

Figure 21. Utilization and imbalance value of the entire data center when PMs=500 and VMs=5000.

C. Comparing energy efficiency

We considered four algorithms here:

1. Round-Robin: The Round-Robin scheduling algorithm, which distributes VM requests to each PM in turn, is the most widely used scheduling algorithm (e.g., by Eucalyptus and Amazon EC2 [18]). The simplicity of this algorithm's implementation is a benefit.
2. MBFD: Modified Best Fit Decreasing. A bin-packing algorithm is MBFD. Best Fit Decreasing is demonstrated to use a maximum of 11/9 optimal solution (OPT)+1 bins (OPT is the optimal solution's maximum allowed number of bins) [6]. To determine which host will result in the least increase in power consumption as a result of this allocation, the MBFD algorithm [6] first ranks all VMs in decreasing order of their current CPU utilizations. This enables the utilisation of resource heterogeneity by prioritising nodes with the lowest power consumption. Because the power boosting is the same for homogeneous resources (PM), the VM can be assigned to any PM that is currently running and still capable of hosting it. The algorithm's allocation phase has a complexity of nm, where n is the total number of virtual machines that must be distributed and m is the total number of hosts. Requests must be sorted for MBFD to be used for offline (or semi-offline) scheduling only.
3. Offline Without Delay (OFWID): OFWID anticipates all requests and executes them precisely and immediately. Before allocating requests to PMs, it arranges requests in ascending order of their IDs and start times. A new PM is activated if none of the currently active PMs can accommodate the request.

Online Without Delay (ONWID): ONWID only ever recognizes a single request. Requests are sent to PMs in ascending order of their IDs. If none of the active PMs can accommodate the request, a new PM is turned on. If there is a predetermined number of PMs and all of them are unable to host the request, the request is blocked.

D. Impact of varying maximum duration of VM requests

Based on Amazon EC2, eight different types of virtual machines (VMs) are considered in this scenario. There are 1000 arrivals (requests) overall, and there are 125 of each sort of VM. The mean interarrival period is set at 5, the maximum intermediate period is set at 50, and the maximum duration of requests is set at 50, 100, 200, 400, and 800 slots, respectively. All requests follow the Poisson arrival process and have exponential service times. Each time slot lasts for five minutes. For instance, if a virtual machine (VM) has 20 slots of required service time, its actual length is 20 × 5, or 100 minutes. The experiments are conducted three times for each set of inputs (requests), and all of the results shown in this chapter are the average of the three runs. Eight different VM types are used to configure PMs, as shown in Table 2. The overall capacity of a VM and a PM in this setup, where there are three different types of PMs (a heterogeneous scenario), is inversely correlated. For purposes of comparison, we assume that every VM is operating at full capacity. While all other parameters remain the same, Figure 22 displays the total energy usage (in kilowatt-hours) of the four methods for maximum durations ranging from 50 to 800 hours.

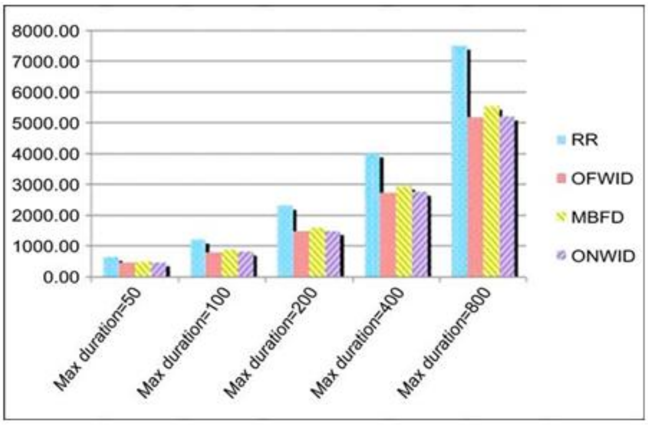

Figure 22. Total energy use (measured in kilowatt hours) when the maximum length of VM requests is changed.

E. Impact of varying the total number of VM requests

The overall number of each sort of PM is then fixed, although the total number of VM requests is varied. The average arrival rate divided by the average service rate is referred to as the system load. Service time has a uniform distribution, while the arrival process has a Poisson distribution. While the overall number of PMs remains fixed at 15, we change the maximum time of each request to increase the system's burden.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

The comparison of overall energy use is shown in Figure 23.

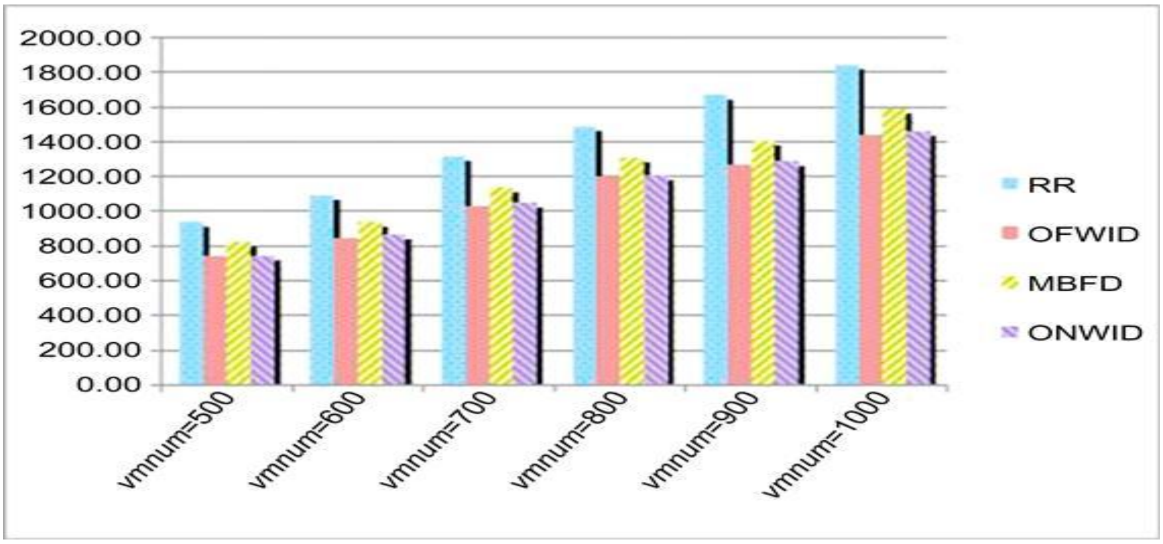

Figure 23. Total energy consumption (in kilowatt hours) varies with the number of VM requests.

VII. CONCLUSION AND FUTURE SCOPE

In this article, we've explained how the use of nature as inspiration can aid in optimising host overload detection and load balancing, and we've also provided a control mechanism for addressing the issue of virtual machine consolidation. The control policy obtained addresses the issue of host overload detection and satisfies the Quality of Service objective for a stationary workload that is well understood and for a specific configuration of state. To assess effectiveness and determine whether all hosts are balanced, we have also suggested the best algorithms for host overload detection. The results of the experimentally conducted investigation are as follows:

a. A method for identifying an overloaded host among a group of several hosts has been demonstrated for the simulated workload and calls for less complex calculations.
b. The Nature Inspiration method helps find the best possible answer by enhancing the level of service directly.
c. The balanced hosts have been placed adjacent to the overloaded and underloaded hosts to achieve a balance between the two.
d. The algorithms suggested allow for the explicit specification of a desired Quality of Service goal, which is achieved by the resulting value of the metric, provided by the system through the offered parameter. The proposed model is based on an algorithm that draws inspiration from nature and requires a few basic assumptions. Nevertheless, experimental research involving an excessive amount of mixed workloads has demonstrated that the method is effective in managing them [13] and [14]. The method performed as well as the best offline algorithm under the simulated workload, which is excellent for an online approach.

ACKNOWLEDGEMENTS

In this paper, we presented CloudSched, a simple cloud resource scheduling emulator. Details about its primary features, design, and implementation are offered. The outcomes of simulations for load balancing and energy-efficient methods are discussed. Developers can find and investigate suitable solutions by using CloudSched while taking into account various resource scheduling strategies and algorithms. We will soon create new indices to assess the effectiveness of associated algorithms for various scheduling schemes, such as multidimensional resource maximization. Additional simulation data are also gathered using variables that include the probability of each VM request, a fixed total number of physical servers, and a variable number of VMs. Different scheduling techniques are currently being contrasted within a CDC, but they can be readily expanded across multiple data centres. The purpose of CloudSched is to compare various IaaS resource scheduling strategies. The system needs to be developed to represent and compare features in SaaS (Software as a Service), PaaS (Platform as a Service), and other areas.

DECLARATION

Funding/ Grants/ Financial Support	No, I did not receive.
Conflicts of Interest/ Competing Interests	No conflicts of interest to the best of our knowledge.
Ethical Approval and Consent to Participate	No, the article does not require ethical approval or consent to participate, as it presents evidence that is not subject to interpretation.
Availability of Data and Material/ Data Access Statement	Not relevant.
Authors Contributions	I am the sole author of the article.

Retrieval Number:100.1/ijeat.E41820612523
DOI: 10.35940/ijeat.E4182.0612523
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

REFERENCES

1. Armbrust M, Fox A, Griffith R, Joseph A, Katz R, Konwinski A, et al. Above the clouds: a Berkeley view of cloud computing. Technical Report No. UCB/EECS-2009-28. University of California at Berkley, CA; February 10, 2009.
2. Google App Engine, <https://appengine.google.com/> [last accessed 25.03.14].
3. IBM blue cloud, <http://www.ibm.com/grid/> [last accessed 26.03.14].
4. Amazon EC2, <http://aws.amazon.com/ec2/> [last accessed 25.03.14].
5. MicrosoftWindows Azure, <http://www.microsoft.com/windowsazure> [last accessed 26.03.14].
6. Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centres in cloud computing. Accepted by Future Generation Computer Systems; 2012. [CrossRef]
7. Buyya R, Ranjan R, Calheiros RN. Modelling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities. In: Proceedings of the seventh high performance computing and simulation conference (HPCS 2009, ISBN: 978-1-4244-4907-1, IEEE Press, New York, NY), Leipzig, Germany; June 21–24, 2009. [CrossRef]
8. Dumitrescu CL, Foster I. GangSim: a simulator for grid scheduling studies. In: Proceedings of the IEEE international symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, UK; 2005. [CrossRef]
9. Youseff L, Butrico M, Da Silva D. Toward a unified ontology of cloud computing. In: Proceedings of the grid computing environments workshop, GCE'08; 2008. IEEE International Conference on Advanced Information Networking and Applications (AINA 2010), Perth, Australia, 20–23 April 2010. [CrossRef]
10. Buyya R, Murshed M. GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for grid computing. *J Concurrency Comput Pract Exp*. 2002;14 Wiley Press, Nov.-Dec. [CrossRef]
11. Howell F, McNab R. SimJava: a discrete event simulation library for Java. In: Proceedings of the first international conference on web-based modelling and simulation; 1998.
12. Wickremasinghe B, Calheiros RN, Buyya R. CloudAnalyst: a CloudSim-based tool for modelling and analysis of large-scale cloud computing environments. In: Proceedings of the 24th.
13. Wood T, Shenoy P, Venkataramani A, Yousif M. Black-box and grey-box strategies for virtual machine migration. In: Proceedings of the symposium on networked systems design and implementation (NSDI); 2007.
14. Singh A, Korupolu M, Mohapatra D. Server-storage virtualisation: integration and load balancing in data centres. In: Proceedings of the 2008 ACM/IEEE conference on supercomputing; 2008, p. 1–12. [CrossRef]
15. Zhang W. Research and implementation of elastic network service [PhD dissertation]. National University of Defence Technology, China (in Chinese) 2000102353.
16. Zheng H, Zhou L, Wu J. Design and implementation of load balancing in web server cluster system. *J Nanjing University Aeronaut Astronaut*. 2006;38.
17. Economou D, Rivoire S, Kozyrakis C, Ranganathan P. *Full-System power analysis and modelling for server environments, Stanford University 2006 2006; [HP Labs Workshop on Modelling*, Benchmarking, and Simulation (MoBS) June.
18. *Full-system power analysis and modelling for server environments, Stanford University,* 2006.

AUTHOR PROFILE

Sebagenzi Jason pursued a Bachelor of Science from Adventist University of Central Africa (Rwanda) and a Master of Science in Information Technology from Jain University (India) in 2021. He is currently Ph.D. Pursuing a computer science major in cloud computing from Jain University (India) and Dean of the Information Technology Faculty.