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

Power Management as a Service (Pmaas) – An Energy Saving Service by "Auto-Fit VM Placement" Algorithm in Cloud Computing for Maximum Resource Utilization at Minimum Energy Consumption Without Live-Migrating the Virtual Machines by using the Concept of Virtual-Servers

Himadri Biswas, Debabrata Sarddar

Abstract*— Now a day Energy Consumption is one of the most promising fields amongst several computing services of cloud computing. A maximum amount of Power resources are absorbed by the data centre because of huge amount of data processing which is increased abnormally. So it's the time to think about the energy consumption in cloud environment. Existing Energy Consumption systems are limited in terms of virtualization because improper virtualization leads to loads imbalance and excessive power consumption and inefficiency in terms of computational power. Billing[1,2] is another exciting feature that is closely related to energy consumption, because higher or lesser billing depends on energy consumption somehow-as we know that cloud providers allow cloud users to access resources as pay-per-use, so these resources need to be optimally selected to process the user request to maximize user satisfaction in the distributed virtualized environment. There may be an inequity between the actual power consumption by the users and the provided billing records by the providers, So any false accusation that may claimed by each other to get illegal compensations. To avoid such accusation, we propose a work to consolidate the VMs using the Power Management as a Service (PMaaS) model in such a way, to reduce power consumption by maximum resource utilization without live-migration of the virtual machines by using the concept of Virtual Servers. The proposed PMaaS model uses a new "Auto-fit VM placement algorithm", which computes tasks resource demands, models a Virtual Machine that fits those demands, and places the Virtual Machines on a Virtual server made by the collective resources (CPU, Memory, Storage and Bandwidth) from the respective schedulers directly connected to the actual physical servers and that has the minimum remaining resources which is large enough to accommodate such a Virtual Machine.*

Revised Manuscript Received on December 30, 2019.

* Correspondence Author

Himadri Biswas*, Department of Computer Science and Engineering, University of Kalyani, Kalyani, Nadia, W.B., India. Email: mr.himadri.biswas@gmail.com

Dr. Debabrata Sarddar, Department of Computer Science and Engineering, University of Kalyani, Kalyani, Nadia, W.B., India. Email:dsarddar1@gmail.com

Keywords: *Power Management as a service, Power Management Service Provider, Scheduler, Task Administrator, Virtual Machine Administrator, Virtual Server Administrator.*

I. INTRODUCTION

Cloud Computing is an exciting feature in today's Technology world, where virtualization is key to the maximization of resource utilization. Without virtualization, all devices need the same energy, emit the same heat, need the same physical space, set-up costs, overhead maintenance, overhead support, equipment costs etc. are directly proportional to the number of machines. Live VM migration [38] is a useful tool in cloud computing, but the drawback is that, in Downtime, the VM service is not available. Energy consumption is closely related to work-load [13] which again related to virtualization. So, Load balancing [14] is very much essential. Load Balancing goes in two directions: Task Scheduling and Virtual Machine (VM) placement [15] [16] [17] [18] [19] [20] [21]. Some approaches for solving VM placement problem are: Constraint Programming; Stochastic Integer Programming; Bin Parking; and Genetic Algorithms [18]. As a solution to the above problems, our proposed model allows the services to be consolidated to a lesser number of physical servers than was initially required. In this model, Cloud infrastructure for PMaaS is solely managed by the Power Management Service Provider (PMSP) without any intervention of the Authorized users (AU), Data Owners (DW), as well as the Cloud Service Providers (CSP). So all needs to be safeguard from any false accusation that may be claimed by each other to get illegal compensations.

II. RELATED WORKS

Virtualization is the core principle of cloud computing, where one computer hosts the presence of multiple computers [3].

Resource sharing [4] offers multi-advantages. A Program which allows multiple operating systems to share a single host is called Hypervisor which ensures that the guest operating systems (called virtual machines) cannot interrupt each other [5]. In [6] the number of physical machines required to deploy the requested virtual machine instances is reduced by combining time series forecasting techniques and bin packing heuristic, but the model has not incorporated multiple resource relationships such as CPU and I/O. In [7] VM placement algorithms use the behavior of VMs to have some properties in general. In [8] a two-level control management system is used to place virtual machines on physical machines and uses combination and multi-phase efficiency to resolve potentially inconsistent scheduling constraints. In [9], VM scheduling constraints are considered to be a single dimension in a multidimensional Knapsack problem. In [10][11] the algorithm is a meta-heuristic one and inspired by the experience of the real ant colonies and is based on their collective foraging behavior. In [12] for balanced resource utilization Max-BRU algorithm is proposed which considers a limited number of resource types, resulting in unbalanced loading or in unnecessary activation of physical servers [12]. Energy-efficient load balancing firefly algorithm [22] claims that many algorithms neglect the efficient use of the exploitation function. In [23], Heuristics has been used as a common approach between systems to enable load balancing among physical servers. In [24], performance variations have been identified and monitored on a physical server hosting VMs. A few basic VM placement algorithms, such as time-shared and space-shared, were presented and compared in [25]. In [26] some techniques for assigning and migrating virtual machines and proposed some migration techniques and algorithms based on the level of server load imbalance. In [27] the most appropriate load-balance scheduling algorithms for traditional web servers have been evaluated. A new Vector Dot load balancing algorithm has been introduced in [28] to work with structured and multi-dimensional resource limitations by taking into account servers and cloud storage. A countable load imbalance measure for virtualized data center servers has been proposed in[29]. In [30] database consolidation was considered a bin packing problem and proposed a VM based algorithm that considers the collective resource demand of the host where the VM is to be placed. An overloaded resource-based approach to VM placement has been presented in [31]. The comparison of different VM scheduling algorithms was presented in [32] and suggested the need for a new efficient VM placement algorithm. In [33] the scheduling algorithm for virtual machines was introduced on the basis of user constraints and multi-dimensional host load. A genetic simulated annealing algorithm for optimizing task scheduling in cloud computing has been proposed and implemented in [34]. In [35][36], a grouping-based genetic algorithm was used to achieve better results than conventional methods and universal heuristic algorithms. VMs must be distributed efficiently in such a way that no device or request fails to respond from the cloud [37]. The primary objective of the VM placement task is to maximize the use of the available resources. Previously, when the number of VMs and PMS was small, mapping of VMs to appropriate PMs was possible manually. However, the current scenario has completely changed that the automation of the placement task is mandatory due to an abnormal increase in the number of VMs and PMs.

In addition, the following issues have been identified in the existing solutions-

First, most of the existing algorithms use a single dimension during VM placement. Nevertheless, the new complex environment specification involves multiple dimensions.

Second, the mapping function of VM-PM is not usually tailored to the resource demand information during scheduling.

Finally, it is very difficult to determine which VMs to place on which PMs have an impact on the performance of the system, which is still an open research issue.

Later, Safety as a Service (SFaaS) [39] and Verification as a Service (VaaS) [2] model have been proposed where Security Service Provider (SSP) and Verification Service Provider (VSP) acts likely as a TTP except that users and the CSPs no one can interact each other directly without the permission of SSP and VSP respectively. So, as and when required this model provides the updated information (consumed resources and storage spaces) to the user, though **Power Management services** is the missing link for virtualization and load balancing the servers.

III. OVERVIEW AND RATIONALE

A. Cloud Service Models

The cloud computing service models are –

Software as a Service (SaaS)

A readymade application, along with any necessary software, operating system, hardware, and network are provided by this SaaS model.

Platform as a Service (PaaS)

Hardware, network and an operating system are provided by this PaaS model, whereas the consumer can install or builds up its own software and applications.

Infrastructure as a Service (IaaS)

Only the hardware and networks are supplied by this IaaS model, the customer set up or develops its own operating systems, software and applications.

B. Proposed Service Model

Fig. 1.Cloud Service Models

Power Management as a Service (PMaaS)

PMaaS under Private Cloud reduces power consumption by using the Virtual Servers concept to maximize resource utilization without a live migration of virtual machines. The proposed PMaaS model uses a new virtual machine placement algorithm-"Auto-fit VM Placement algorithm.

The main concept is to map the VMs to the actual physical resources without any partiality either or both sides and no chance of intervention between the users, the Data Owners (DWs) and the Service Providers directly. Because once a user, the DWs or a service provider registered in PMaaS model, has to go through this service.

C. Power Management Service Provider (PMSP):

PMSP under PMaaS, plays the important role of VM-VS mapping. It also plays the important role to establish a smooth communication between the users, the data owners and the CSP. So before allocating the resources on the cloud server, PMSP registers the Users, the Data Owners and the CSP's to avoid future disputes. Hence, PMSP maintains a log table which holds all identification keys of the users, Data Owners and their respective CSPs. PMSP is deeply related with the following modules:

D. Task Administrator (TA):

Task Administrator group the Tasks (submitted by the user) and create the VMs with maximum task requirements for each group.

E. Virtual Machine Administrator (VMA):

The task of VMA is to Schedule the VMs for allocating resources to each.

F. Virtual Server Administrator (VSA):

VSA is engaged to create and manage the Virtual Servers (VSs) based on the available resources from respective scheduler.

G. CPU Scheduler (CS):

It arranges the Physical Machines (PMs) in ascending order of CPU availability.

H. Memory Scheduler (MS):

It arranges the Physical Machines (PMs) in ascending order of RAM availability.

I. Storage Scheduler (SC):

It arranges the Physical Machines (PMs) in ascending order of Disk availability.

IV. PROPOSED WORK

Emerging cloud computing infrastructures provide pay-based computing resources as needed. Maximum resource utilization using less power consumption is therefore one of the most challenging features to improve cloud usability. Figure 2 describes our proposed service model "**PMaaS**", where PMSP helps to map the VMs to PMs in such a way that resource utilization should be maximum at minimum energy consumption. As we know that there are different cloud service models and their intended service providers available in the market, but most of them do not allow customers to verify their used resources or allowed on request basis. Once allowed, most of the cases they do not realize their return records are correct or not. In this situation they need help of a third party on faith. Our aim in this work is to provide a service model PMaaS where the users if they want must be going through this model before getting any services from the CSP's and in that case no CSPs should provide any final intimation to the users without getting approval from PMSP under PMaaS.

Figure 3 describes the working model of PMSP, where either DWs or Users or CSPs whatever may be before using the resources take the permission from PMSP. After receiving the request PMSP first checks the authorization from its log table, then allowing them to allocate the resource using its Resource Management Manager (RMM).

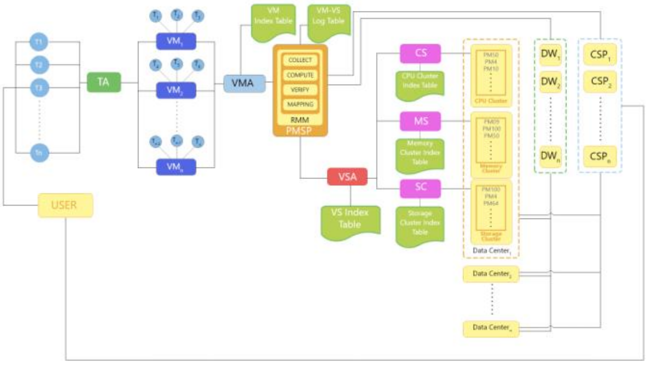

Fig. 2.Proposed Architecture of Power Management as a Service (PMaaS) Model

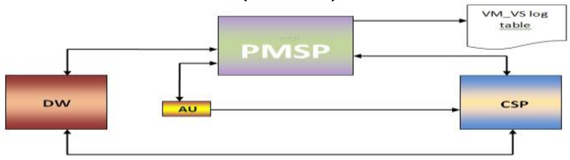

Fig. 3.PMSP offering PMaaS

A. Working procedure of PMaaS Model

In the above figure 2 a user requests for services the CSP checks for authentication and SLA before determining whether to accept or reject the request and forwards it to the Task Administrator (TA) as well as the PMSP. Upon receiving the tasks from the user TA schedule and group the tasks in a FCFS basis. TA chooses the maximum resource requirements for each group as model the VM and sends the VMS with maximum task requirements to the Virtual Machine Administrator (VMA), where VMA maintain a queue of VMs as the order sent by the TA. Now VMA request to PMSP for allocating the resources of VM. On the other side, CSP forwards the request to PMSP for getting the service of its intended users. Now, PMSP checks for CSP's authentication and searches for the availability of the Data Owner (DW). Upon receiving the request from VMA, PMSP forwards the request to Virtual Server Administrator (VSA) for getting the PM(s) for allocating the VM(s). Here VSA is engaged to manage and create the Virtual Servers (VS) based on the available resources from respective schedulers- CPU Scheduler (CS) which arranges the PM(s) in ascending order of CPU core availability, Memory Scheduler (MS) which arranges the PM(s) in ascending order of RAM availability, Storage Scheduler (SS) which arranges the PM(s) in ascending order of Disk availability. Before allocating the VMs, VSA searches the required CPU Core from its CPU scheduler which holds the PMs in ascending order of their CPU core availability. The same thing can be happened to other schedulers. Though all the schedulers contain the available resources in ascending order, so when one VM is selected to allocate the required resources (CPU, Memory, Storage) to the respective schedulers, it will search from the beginning of the respective scheduler arrays- if the required resources of the VM is less than or equal to the first element of the respective scheduler arrays, it will be selected for the VM placement. In case of less than i.e., if the required resources of the VM is less than the first element of the respective scheduler arrays, then the equal amount of the required resource will be granted for VM placement and the remaining amount of actual resource will be rearranged in ascending order. The searching process continues from the beginning to the end of each scheduler until the exact matching is found. Our new "Auto-fit" Algorithm performs this automatic placement of VMs to the PMs and obviously it will allocate the minimum amount of required resources. Once allocate the resources by the respective VM, one VS will be created and all the schedulers again rearranged by its available resources. Now the newly created VS forwarded by the VSA to the PMSP for approval and assigning the corresponding VM on it. From collecting to computing, verifying and mapping the VMs to the PMs done by the Resource Management Manager (RMM) of PMSP. So, it is very clear that without the PMSP's permission either the DWs' or the CSPs' no one can directly interacts with each other for resource allocation. In case of any dispute, if required PMSP obtains the detailed records of the alleged Customers, Data Owners or CSPs from its own VM-VS Mapping log table. So there is no chance of mistake or cheating by the DWs' or the CSPs' to the Cloud Users.

B. Components of PMaaS

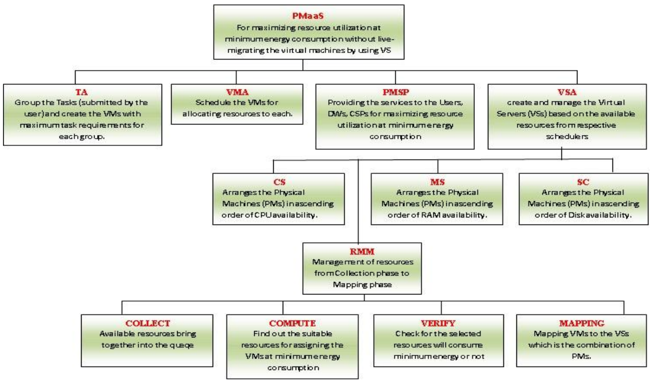

Fig. 4.Components of PMaaS

V. ALGORITHM, FLOWCHART, PSEUDO CODE

A. Auto-fit VM placement Algorithm

1. User request for services
2. CSP checks for user's authentication.
3. If the user is authentic—
   - 3.1 User submits the tasks to TA.
     - 3.1.1 TA schedules the tasks in a FCFS basis
     - 3.1.2 TA group the tasks
     - 3.1.3 TA chooses the maximum resource requirements for each group as model the VM
     - 3.1.4 TA sends the VMs (with maximum task requirements) to the VMA.
     - 3.1.5 VMA maintain a queue of VMs in the order sent by TA of CPU requirement for each VM.
     - 3.1.6 VMA request for resources to PMSP.
   - 3.2 CSP forwards the requests to PMSP.
     - 3.2.1 PMSP maintains an index table of CSPs and the corresponding DWs.
     - 3.2.2 PMSP searches for the respective DW.
     - 3.2.3 If matches—
       - 3.3.3.1 CSP accepted for services.
     - 3.2.4 Else—
       - 3.2.4.1 Go to step 3.2
   - 3.3 PMSP forwards the request to VSA.
     - 3.3.1 VSA receives the periodic signal for available resources from respective schedulers.
     - 3.3.2 VSA checks for resource availability from its resource table.
     - 3.3.3 If available—
       - 3.3.3.1 Create VSs for assigning the VMs.
       - 3.3.3.2 VSA forwards the VSs to the PMSP.
       - 3.3.3.3 PMSP approves the VS and assign the corresponding VM on it.
       - 3.3.3.4 PMSP maintain s a log table for VM-VS mapping.
4. Else –
   - 4.1 "User not accepted for services" – message return to the user, i.e., go to step 1.
5. End.

B. Flowchart :

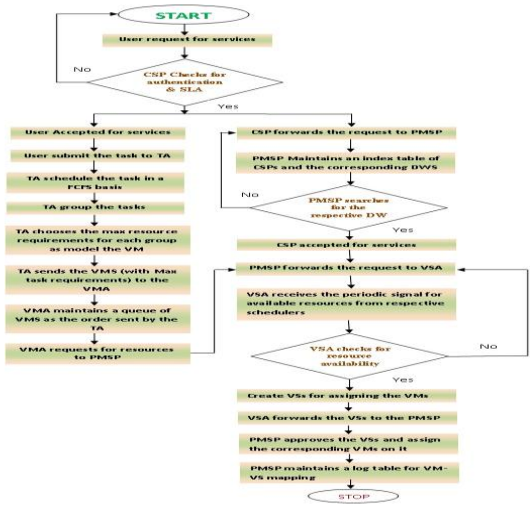

Fig. 5.Flowchart of PMaaS

C. Pseudo code (VM-VS Mapping)

Notations:

AWL: Average work load of an instance

PS :Processor speed of an instance (amount of data processing per second)

Th : Threshold value of an instance
Ru : Resource used by and instance
T : Set of tasks
P : set of processes
VM : set of virtual machines
PM : set of physical machines
VS : set of virtual servers
VMmap : set of virtual machines which are mapped
PMnotmap: set of physical machines on which VM can't be mapped
So : initial state
PMcap : capacity of each PM in an instance
CS : CPU Scheduler-set of PMs with available CPU cores stored in ascending order of cores
MS : Memory Scheduler- set of PMs with available RAMs stored in ascending order of RAMs
SS : Storage Scheduler- set of PMs with available Disk spaces stored in ascending order of Disk spaces
PMcpucap : Total CPU capacity of a PM
PMmemcap: Total RAM capacity of a PM
PMstorcap : Total Disk spaces of a PM
MTR : Maximum Task Requirement
p : Number of process
n : Number of VMs
k : Number of tasks of a process
s : Number of VSs
r : number of PMs
RA : Resource Allocation

```
Begin CREATE_VM
Data:   P initialize to zero
Data:   VM initialized to zero
Data:   i initialized to zero
Data:   set _of_process
Data:   cpu_requirement initialized to zero
Data:   mem_requirement initialized to zero
Data:   stor_requirement initialized to zero
Data:   MTR initialized to zero
Data:   p initialized to zero
Data:   n initialized to zero

While true do
      For each set_of_process  [i=1 to p] do
      cpu_requirement← Max_cpu_req(p[i])
      mem_requiremen←Max_mem_req(p[i])
      stor_requirement←Max_stor_req(p[i])
      MTR←Max_Task_req(cpu_requirement,
                mem_requirement, stor_requirement)
      Store(VM[i], MTR)
      i←i+1
      n←i
      Endfor
Endwhile
End CREATE_VM

Begin INITIATE_PROCESS
Data:   P initialize to zero
Data:   i initialized to zero
Data:   j initialized to zero
Data:   k initialized to zero
Data:   Cn- need of CPU initialized to zero
Data:   Mn- need of RAM initialized to zero
Data:   Sn- need of Disks initialized to zero
Data:   RESn- need of total resources initialized to zero

While true do
      For each process [i=1 to p] do
         For each task [j=1 to k] do
            Cn← cpu_need(T[j])
            Mn←mem_need(T[j])
            Sn←stor_need(T[j])
            RESn← resource_need(Cn, Mn, Sn)
            Store(P[i] T[j], RESn)
         j←j+1
         Endfor
      i←i+1
      Endfor
Endwhile
End INITIATE_PROCESS

Begin CREATE_VS
Data:   resource initialize to zero
Data:   get_resource initialize to zero
Data:   set_resource initialized to zero
Data:   verified_resource initialized to zero
Data:   allocate_resource initialized to zero
Data:   s initialized to zero
While true do
For each VM  [i=1 to n] do
get_resource ← RMM.COLLECT(Resource)
set_resource← RMM.COMPUTE(get_resource)
verified_resource←RMM.VERIFY(set_resource)
allocate_resource←RMM.MAPPING(verified_resource)
CS←sort_asc_cpu_avl {PMi_cpu_avl}; for PMi, 1≤i≤r
MS←sort_asc_memory_avl {PMi_mem_avl}; for PMi, 1≤i≤r
SS←sort_asc_storage_avl {PMi_storage_avl}; for PMi, 1≤i≤r
Store(VS[s], allocate_resource)
s←s+1
Endfor
Endwhile
End CREATE_VS

//For INITIATE_PROCESS
Begin cpu_need
Data:   T_id – Task identification
Data:   set_Cn – Setup CPU need
 set_Cn← TA.set_cpu_need(T_id)
Return set_Cn
End cpu_need

Begin mem_need
Data:   T_id - Task identification
Data :  set_Mn – Setup Memory need
set_Mn←TA.set_mem_need(T_id)
Return set_Mn
End mem_need
Begin stor_need
Data:   T_id - Task identification
Data:   set_Sn - Setup storage need
set_Sn←TA.set_stor_need(T_id)
return set_Sn
End stor_need
```

Begin **resource_need**
Data: **set_Rn** – Setup resource need
set_Rn← TA.set_resource_need(Set_Cn, Set_Mn, Set_Sn)
return Set_Rn
End **resource_need**

// for CREATE_VM
Begin m**ax_cpu_req**
Data: **T_id**
Data: **P_id**
Data: **set_cpu_req**
Set_cpu_req←TA.set_max_cpu_req(P_id)
Store ([P_id][T_id], set_cpu_req)
return ([P_id][T_id])
End **max_cpu_req**

Begin **max_memory_req**
Data: **T_id** - Task identification
Data: **P_id** – Process identification
Data: **Set_memory_req**
Set_memory_req←TA.set_max_memory_req(P_id)
Store ([P_id][T_id], set_memory_req)
return ([P_id][T_id])
End **max_memory_req**

Begin **max_storage_req**
Data: **T_id** - Task identification
Data: **P_id** – Process identification
Data: **Set_storage_req**
Set_storage_req←PMSP.VMA.TA.set_max_storage_req(P_id)
Store ([P_id][T_id], set_storage_req)
return ([P_id][T_id])
End **max_storage_req**

Begin **max_task_req**
Data: **T_id** - Task identification
Data: **P_id** – Process identification
Data: **Set_MTR –** Set maximum task requirement
Set_MTR← TA.set_max_task_req(([P_id][T_id], Set_cpu_req), ([P_id][T_id], Set_memory_req), ([P_id][T_id], Set_storage_req))
Store ([VM_id], set_MTR)
return ([VM_id])
End **max_task_req**

// for CREATE_VS
Begin **COLLECT**
Data**: Virtual Server, VS**←{T, P, VM, PM, VS, P`, VM`, PM`}
Where, set_of_tasks – T
Set_of_processes – P
Set_of_virtual_machines – VM
Set_of_virtual_servers – VS
Set_of_machine which are mapped – VM`
Set_physical_machines on which VMs can't be mapped – PM`
Data: **Set_of_process, P** ←{P0, P1, P2, ..., Pp}
Where, P0 ← {T0, T1, T2, ..., Tk}
P1← {T0, T1, T2, ..., Tk}
....
Pp ← {T0, T1, T2, ..., Tk}
Where, k – number of tasks of a process
p – number of processes submitted by the user
Data: **Set_of_VMs, VM** = {VM0, VM1, VM2, ..., VMn}
Data: **Set_of_PMs, PM** = {PM0, PM1, PM2, ..., PMr}
Where, n – number of VMs to be mapped
r – number of PMs
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage}
Where, VMi_cores is the number of CPU cores required by VM[i]
VMi_memory is the amount of RAM required by VM[i]
VMi_storage is the amount of disk space required by VM[i]
Data: **PM[i]** = {PMi_cores, PMi_memory, PMi_storage}
Where, PMi_cores is the number of CPU cores available in PM[i]
PMi_memory is the amount of RAM available in PM[i]
PMi_storage is the amount of disk space available in PM[i]
Data: **VS[i]** = $\{CS_{PMi\_cores \geq VMi\_cores}, MS_{PMi\_memory \geq VMi\_memory}, SS_{PMi\_storage \geq VMi\_storage}\}$; for PMi, $1 \leq i \leq r$ & VMi, $1 \leq i \leq n$
Data: **Resource**
Resource←(CS.select_cpu,MS.select_memory,SS.select_storage)
Return resource
End **COLLECT**

Begin **select_cpu**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage};
Data: **CS** = {PMi_cpu_avl}; for PMi, 1≤i≤r
For each CS [j=1 to r] do
if (CS[j] ≥ VMi_cores)
Return CS[j]
Endif
j←j+1
End For
End **select_cpu**

Begin **select_memory**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage};
Data: **MS** = {PMi_memory_avl}; for PMi, 1≤i≤r
For each MS [j=1 to r] do
if (MS[j] ≥ VMi_memory)
Return MS[j]
Endif
j←j+1
End For
end **select_memory**

Begin **select_storage**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage};
Data: **SS** = {PMi_storage_avl}; for PMi, 1≤i≤r
For each SS [j=1 to r] do
if (SS[j] ≥ VMi_storage)
Return SS[j]
Endif
j←j+1
End For
End **select_storage**
Begin **COMPUTE**
Data: PMi_Ravl : Resource available of each PM
PMi_Th: Threshold value of PMi
PMi_Ru : Resource used by PMi
AWL: Average work load of an instance
PS: Processor speed of an instance (amount of data processing per second)
Ru: Resource used by all the PMs of a Data Centre

Ravl: Resource available of all the PMs of a Data Centre
get_resource
While true do
PMi_Th← AWL * PS
PMi_Ravl ← PMi_Th – PMi_Ru
PMi_Ravl←PMi_Th– (PMi_cpu_used + PMi_memory_used + PMi_storage_used)

$$Ru \leftarrow \{\sum_{i=1}^{r} PMi\_cpu\_used, \sum_{i=1}^{r} PMi\_memory\_used, \sum_{i=1}^{r} PMi\_storage\_used\}$$

$$\text{so, } \sum_{i=1}^{r} PMi\_cpu\_avl \leftarrow \sum_{i=1}^{r}(PMi\_Th - PMi\_cpu\_used)$$

$$\sum_{i=1}^{r} PMi\_memory\_avl \leftarrow \sum_{i=1}^{r}(PMi\_Th - PMi\_memory\_used) \ \&$$

$$\sum_{i=1}^{r}(PMi\_storage\_avl \leftarrow \sum_{i=1}^{r}(PMi\_Th - PMi\_storage\_used)$$

$$\text{Therefore, } Ravl \leftarrow \{\sum_{i=1}^{r} PMi\_cpu\_avl), \sum_{i=1}^{r}(PMi\_memory\_avl), \sum_{i=1}^{r}(PMi\_storage\_avl)\}$$

← {Capacity(CS), Capacity (MS), Capacity (SS)}

get_resource←Ravl
return get_resource
End **COMPUTE**

Begin VERIFY
Data: set_resource
Begin **verify_cpu**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage}
Data: **VM`[i]** = {VM`i_cores, VM`i_memory,VM`i_storage}
Data: **CS** = {PMi_cpu_avl}; for PMi, 1≤i≤r
For each CS [j=1 to r] do
    if (CS[j] ≥ VMi_cores)
        VM`i_cores←VMi_cores
    Endif
j←j+1
End For
End **verify_cpu**

Begin **verify_memory**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage}
Data: **VM`[i]** = {VM`i_cores, VM`i_memory, VM`i_storage}
Data: **MS** = {PMi_ memory_avl }; for PMi, 1≤i≤r
For each MS [j=1 to r] do
    if (MS[j] ≥ VMi_ memory)
        VM`i_ memory ←VMi_ memory)
    Endif
j←j+1
End For
end **verify_memory**

Begin **verify_storage**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage}
Data: **VM`[i]** = {VM`i_cores, VM`i_memory,VM`i_storage}
Data: **SS** = {PMi_ storage_avl }; for PMi, 1≤i≤r
For each SS [j=1 to r] do
    if (SS[j] ≥ VMi_ storage)
        VM`i_ storage ←VMi_ storage
    Endif
j←j+1
End For
end **verify_storage**
set_resource← (VM`i_cores, VM`i_memory, VM`i_storage)
return resource
End VERIFY

Begin MAPPING
Data: verified_resource
Begin **mapping_cpu**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage}
Data: **VM`[i]** = {VM`i_cores, VM`i_memory,VM`i_storage}
Data: **CS** = {PMi_cpu_avl}; for PMi, 1≤i≤r
For each CS [j=1 to r] do
    CS[j]←VM`i_cores
    VM←(VM-VMi_cores)
    CS←(CS-(PMj_cpu_avl –VMi_cores))
j←j+1
End For
end **mapping_cpu**

Begin **mapping_memory**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage}
Data: **VM`[i]** = {VM`i_cores, VM`i_memory,VM`i_storage}
Data: **MS** = {PMi_memory_avl}; for PMi, 1≤i≤r
For each MS [j=1 to r] do
    MS[j]← VM`i_memory
    VM←(VM- VMi_memory)
    MS←(MS-(PMj_ memory_avl – VMi_memory))
    Endif
j←j+1
End For
end **mapping_memory**

Begin **mapping_storage**
Data: **VM[i]** = {VMi_cores, VMi_memory, VMi_storage}
Data: **VM`[i]** = {VM`i_cores, VM`i_memory,VM`i_storage}
Data:**SS** = {PMi_storage_avl}; for PMi, 1≤i≤r
For each SS [j=1 to r] do
    SS[j]← VM`i_storage
    VM←(VM- VMi_storage)
    SS←(SS-(PMj_ storage_avl – VMi_storage))
    Endif
j←j+1
End For
End **mapping_storage**
verified_resource ←(CS[j],

MS[j], SS[j])
return verified_resource
End MAPPING

VI. DETAILED ANALYSIS OF PMAAS MODEL (A CASE STUDY)

Our Auto-Fit VM Placement Algorithm is an automatic VM placement algorithm. This algorithm uses the maximum task requirements, such as CPU speed, RAM capacity, DISK size, and NET bandwidth, to model a VM for the execution of all other tasks in a group. For the simplicity here we only use the CPU*speed*, RAM*capacity* and DISK*size*. For each newly created VM, it gets all activated Physical Machines with available CPU cores, RAM and Disks from the respective Clusters like CPU scheduler, Memory scheduler and Storage scheduler and makes the Virtual Servers (VS) for each activated VM that has the least available resources that can match the requirements of the VM and then assigns the VM to such a VS. If no PM has resources to match the VM's requirements, a new PM is activated and the process starts all over again. As it is difficult to access real data centers or cloud infrastructures, we used simulation-based tests that can be easily reproducible to compare the performance of the proposed algorithm with the existing works currently being used by most cloud service providers.

A. Illustrative Example

Table I: Tasks with their resource requirements

Task_id	CPU (Cores)	RAM (GB)	Disk (GB)
T1	1	8	25
T2	2	4	40
T3	4	6	100
T4	1	4	30
T5	1	3	50
T6	2	2	35
T7	4	3	200
T8	6	2	60
T9	5	4	100
-----	-----	------	-----

Group the tasks and choose the maximum task requirements for each group to create the VM

Table II: VM creation based on maximum task requirement

VM_id	CPU (Cores)	RAM (GB)	Disk (GB)
VM1	4	8	100
VM2	2	4	50
VM3	6	4	200
-----	-----	------	-----

To explain our algorithm, assume that we have nine tasks (T1 ... T9) with resource requirements in Table I, then group the tasks (three tasks in a group) and choose the maximum task requirements for each category to construct the VM, i.e. three VMs (VM1 ... VM3) shown in Table II.

Here, we compare our proposed algorithm with two existing (First-fit and Best-fit) algorithms using the following experimental setup, and then show that our proposed algorithm would produce a better result.

B. Experimental Setup

To illustrate our example, assume that we have three PMs with available resources and 11 VM requests with different resource requirements as shown in Table III and Table IV, respectively, where the VMs in Table IV will be allocated to the available PMs as shown in Table III. Figure 6 and Figure 7 depicting the VM placement process by the First fit and Best-fit algorithm while Table V depicting the VM-VS mapping table of the proposed "Auto-fit VM Placement" algorithm.

Table III: Available resources of PMs

PM_id	CPU (cores)	RAM (GB)	Disk (GB)
PM1	32	48	600
PM2	24	32	800
PM3	16	32	600

Table IV: Virtual Machine Requirements

VM_id	CPU (cores)	RAM (GB)	Disk (GB)
VM1	4	8	100
VM2	2	4	50
VM3	6	4	200
VM4	8	4	100
VM5	6	12	100
VM6	4	4	200
VM7	8	4	100
VM8	4	8	200
VM9	4	16	100
VM10	12	6	200
VM11	10	28	250

PM1

Available Resources after assigning VMs	CPU (cores) 32	RAM (GB) 48	DISK (GB) 600
	6	16	50
VM5	6	12	100
VM4	8	4	100
VM3	6	4	200
VM2	2	4	50
VM1	4	8	100

PM2

Available Resources after assigning VMs	CPU (cores) 24	RAM (GB) 32	DISK (GB) 800
	4	0	200
VM9	4	16	100
VM8	4	8	200
VM7	8	4	100
VM6	4	4	200

PM3

Available Resourc-es after assigning VMs	CPU (cores) 16	RAM (GB) 32	DISK (GB) 600
	4	26	400
VM10	12	6	200

Fig. 6.VM allocation based on First-fit algorithm

PM1

Available Resources after assigning VMs	CPU (cores) 32	RAM (GB) 48	DISK (GB) 600
	0	0	50
VM9	4	16	100
VM7	8	4	100
VM5	6	12	100
VM4	8	4	100
VM2	2	4	50
VM1	4	8	100

PM2

Available Resourc-es after assigning VMs	CPU (cores) 24	RAM (GB) 32	DISK (GB) 800
	2	18	200
VM10	12	6	200
VM6	4	4	200
VM3	6	4	200

PM3

Available Resourc-es after assigning VMs	CPU (cores) 16	RAM (GB) 32	DISK (GB) 600
	12	24	400
VM10	4	8	200

Fig. 7.VM allocation based on Best-fit algorithm

Table V: VM-VS Mapping table

List of VMs to be assigned				CPU Cluster			Memory Cluster			Storage Cluster			List of VSs with assigned PMs			
VM_id	CPU (cores)	RAM (GB)	Disk (GB)	PM_id	CPU (Cores) Avail-able	CPU (Cores) Assig-ned	PM_id	RAM (GB) Avail-able	RAM (GB) Assig-ned	PM_id	Disk (GB) Avai-lable	Disk (GB) Assig-ned	VS_id	PM_id (from CPU cluster)	PM_id (from Memory cluster)	PM_id (from Storage cluster)
VM1	**4**	**8**	**100**	**PM3**	**16**	**4**	**PM2**	**32**	**8**	**PM1**	**600**	**100**	**VS1**	**PM3**	**PM2**	**PM1**
VM2	2	4	50	PM3	12	2	PM2	24	4	PM1	500	50	VS2	PM3	PM2	PM1
VM3	6	4	200	PM3	10 4	6	PM2	20	4	PM1	450	200	VS3	PM3	PM2	PM1
VM4	8	4	100	PM2	24	8	PM2	16	4	PM1	250	100	VS4	PM2	PM2	PM1
VM5	6	12	100	PM2	16 10	6	PM2	12 0	12	PM1	150 50	100	VS5	PM2	PM2	PM1
VM6	4	4	200	PM3	4 0	4	PM3	32	4	PM3	600	200	VS6	PM3	PM3	PM3
VM7	8	4	100	PM2	10 2	8	PM3	28	4	PM3	400	100	VS7	PM2	PM3	PM3
0	4	8	200	PM1	32	4	PM3	24	8	PM3	300	200	VS8	PM1	PM3	PM3
VM9	4	16	100	PM1	28	4	PM3	16 0	16	PM3	100 0	100	VS9	PM1	PM3	PM3
VM10	12	6	200	PM1	24	12	PM1	48	6	PM2	800	200	VS10	PM1	PM1	PM2
VM11	10	28	250	PM1	12 2	10	PM1	42 14	28	PM2	600 350	250	VS11	PM1	PM1	PM2
Available Resources	CPU(cores)				4	RAM		14	Disk		400					

After allocation of ten VMs (VM1 ....VM10) among eleven VMs (in table IV) by using three PMs (in table III) with these three algorithms (First-fit, Best-fit, Auto-fit), the remaining available resources of three PMs in each cases are same i.e., 14 CPU(cores), 42 GB RAM and 650 GB Disk spaces are available.

From Fig.6 and 7 it is clear that First-fit and Best-fit will fail to allocate VM11 to the PMs due to fragmentation, i.e., according to these two algorithms one VM should be allocated to one and only one PM with available resources required by the VM. So, it is not possible to allocate VM11 (10 CPU(cores),28 GB RAM, 250 GB Disk spaces) to any one of the PMs at a time though much higher resources are available by all the PMS. On the other hand, our proposed algorithm will easily allocate VM11 to the PMs because of the concept of Virtual server (VS) which is created by the PMs depends upon the requirements of VMs.

C. Analysis

Table VI and VII representing the available resources and allocated resources by VMs respectively. From figure 8 and 9 it is found that up to 10 number of VMs three algorithm produces the same result, but at the time of allocation of VM11 First-fit and Best-fit fails to meet the criteria, whereas our proposed algorithm works very efficiently such as maximum resources are utilized than other two algorithms. In table VI it is noticed that red colored cells are "zero", i.e., no such resources are available and most of them are allocated by our Auto-fit algorithm. Figure 8 shown that after allocation of VM11, 4 CPU cores, 14 GB RAM and 400 GB Disk spaces are available till now which further may be utilized by other VMs. On the other hand, figure 9 shows that after allocating all the VMs Auto-fit algorithm utilized maximum resources than other algorithms. So the proposed algorithm achieves better placement of VMs, introduces load balancing among all the PMs which tends to the power consumption rate can be minimized at maximum resource utilization with low cost overhead with the concept of Virtual Server.

Table VI: Available resources after allocation of VMs by First-fit, Best-fit & Auto-fit algorithms

VM placement Algorithms	PM1 CPU(CORES)	PM1 RAM(GB)	PM1 DISK(GB)	PM2 CPU(CORES)	PM2 RAM(GB)	PM2 DISK(GB)	PM3 CPU(CORES)	PM3 RAM(GB)	PM3 DISK(GB)	Total available resource CPU(CORES)	Total available resource RAM(GB)	Total available resource DISK(GB)
AUTO-FIT (VM1....VM11)	2	14	50	2	0	350	0	0	0	4	14	400
BEST-FIT (VM1....VM10)	0	0	50	2	18	200	12	24	400	14	42	650
FIRST-FIT (VM1....VM10)	6	16	50	4	0	200	4	26	400	14	42	650
AUTO-FIT(VM1....VM10)	12	42	50	2	0	600	0	0	0	14	42	650

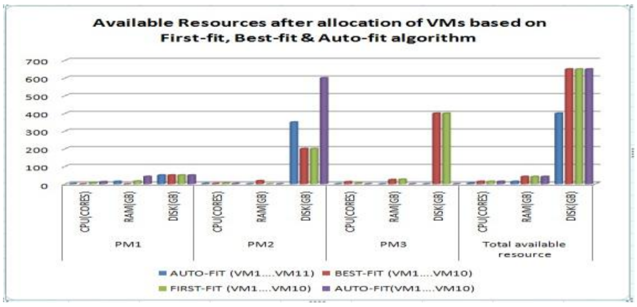

Fig. 8. Resources available after allocation of VMs based on First-fit, Best-fit & Auto-fit algorithms

Table VII: Resource allocation by VMs based on First-fit, Best-fit & Auto-fit algorithms

VM placement Algorithms	PM1 CPU(CORES)	PM1 RAM(GB)	PM1 DISK(GB)	PM2 CPU(CORES)	PM2 RAM(GB)	PM2 DISK(GB)	PM3 CPU(CORES)	PM3 RAM(GB)	PM3 DISK(GB)	Total allocated resources CPU(CORES)	Total allocated resources RAM(GB)	Total allocated resources DISK(GB)
AUTO-FIT (VM1....VM11)	30	34	550	22	32	450	16	32	600	68	98	1600
BEST-FIT (VM1....VM10)	32	48	550	22	14	600	4	8	200	58	70	1350
FIRST-FIT (VM1....VM10)	26	32	550	20	32	600	12	6	200	58	70	1350
AUTO-FIT(VM1....VM10)	20	6	550	22	32	200	16	32	600	58	70	1350

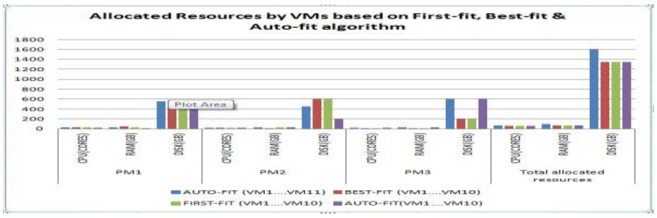

Fig. 9. Resource allocation by VMs based on First-fit, Best-fit & Auto-fit algorithms

VII. CONCLUSION

We have offered our novel approach to PMaaS, which considers VM user constraints along with physical host load factor to address the issue of mapping VMs into PMs in such a way that the number of PMs used is minimized, over-use and under-use of PM resources can be identified and resolved at the same time without violating any SLA agreements. Since we consider this as a Power Management Service that not only minimizes power consumption but serves as a mediator between authorized users, data owners and cloud service providers, i.e. without the permission of the Power Management Service Provider, no one can communicate with each other. It may avoid the inequity between the actual consumption of electricity by the users and the billing records provided by the providers, so avoid any false accusations that may be claimed against each other in order to obtain illegal compensation. Based on the analysis, we have shown that our proposed algorithm uses a minimum number of physical machines to host a set of VMs, which also reduces the power consumption of the data center at a higher resource utilization rate by using a minimum number of physical machines. Another major enrichment in our algorithm is the lower percentage of load imbalance value and the percentage of VMs that breach their SLA.

VIII. FUTURE SCOPE

Our goal is to enrich the QoS, end user utility, satisfactory level of the users as well as the service providers as far as low investment cost is possible and in future we try to implement other Power Management modules to make cloud computing more accessible.

ACKNOWLEDGEMENT

I would like to convey my wholehearted respect and sincere feeling to my revered research adviser, Dr. Debabrata Sarddar, for sparing his precious time and assimilating new, flawless concepts at every stage of our work. Without his kind cooperation, motivation, and careful guidance, I may not have been able to perform this analysis work successfully.

REFERENCES

1. Rajesh Bose, Himadri Biswas, Debabrata Sarddar , Manas Kumar Sanyal, "Cloud Billing & Verification Of Consumed Resources and Storage Spaces by a Cloud User" in International Journal of Applied Engineering Research ISSN 0973-4562 Volume 11, Number 9 (2016) pp 6568-6576.
2. Himadri Biswas, Debabrata Sarddar, "Verification as a Service (VaaS): A Trusted Multipurpose Service for Accounting, Billing Verification and Approval of Consumed Resources at Low Computational Overhead to Enhance the Cloud Usability", International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 19 (2018) pp. 14402-14410 © Research India Publications. http://www.ripublication.com
3. Obasuyi, G.C. and Sari, A. "Security challenges of virtualization hypervisors in virtualized hardware environment", International Journal of Communications, Network and System Sciences, Vol. 8, Pp. 260-273, 2015.
4. Tamane, S. "A review on virtualization : A cloud technology", International Journal on Recent and Innovation Trends in Computing and Communication, Vol. 3, No 7, Pp. 4582-4585, 2015.
5. . Durairaj, M. and Kannan, P. "A study on virtualizaiton techniques and challenges in cloud computing", International Journal of Scientific and Technology Research, Vol. 3, No 11, Pp. 147-151, 2014.
6. Sindelar, M., Sitaraman, R.K., Shenoy, P.: "Sharing-Aware Algorithms for Virtual Machine Colocation". In: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures. San Jose, California, USA, June 2011.
7. Xu, J., Fortes, J.A.B.: "Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments". In: Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing. Hangshou, PR of China, Dec 2010.
8. Singh, A., Korupolu, M., Mohapatra, D.: "Server-Storage Virtualization: Integration and Load Balancing in Data Centers". In: Proc. of the 2008 ACM/IEEE conference on Supercomputing (SC'08). pp. 53:1–53:12. Austin, TX 2008
9. Meng, X., Pappas, V., Zhang, L.: "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement". In: Proceedings of IEEE INFOCOM. San Diego, CA, USA, March 2010.
10. G. Yongqiang, G. Haibing, Q. Zhengwei, H. Yang and L. Liang, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," Journal of Computer and System Sciences, vol. 79, pp. 1230-1242, 2013.
11. J. S. Benita, "Survey on VM Placement Algorithms," International Journal of Engineering Trends and Technology, vol. IV, no. 7, pp. 349-352, 2013
12. T. H. Nguyen, D. F. Mario and Y. J. Antti, "A virtual machine placement algorithm for balanced resource utilization in cloud data centers," in Cloud Computing (CLOUD), Anchorage, 2014.
13. Nikolay Grozev and Rajkumar Buyya, "Performance Modeling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud
14. Environments". The Computer Journal, 2013.
15. K. M. Nitin and M. Nishchol, "Load Balancing Techniques: Need, Objectives and Major Challenges in Cloud Computing - A Systematic Review," International Journal of Computer Applications, vol. 131, pp. 11-19, 2015.
16. E. Barlaskar, Y. J. Singh and B. Isaac, "Energy-efficient virtual machine placement using enhanced firefly algorithm," Multiagent and Grid Systems, vol. 12, no. 3, pp. 167-198, 2016.
17. J. S. Benita, "Survey on VM Placement Algorithms," International Journal of Engineering Trends and Technology, vol. IV, no. 7, pp. 349-352, 2013.
18. Z. Linquan, Y. Xunrui, L. Zongpeng and W. Chuan, "Hierarchical Virtual Machine Placement in Modular Data Centers," in IEEE 8th International Conference on Cloud Computing, 2015.
19. H. K. Md, C. S. Gholamali and G. Sudhakar, "VM Placement Algorithms for Hierarchical Cloud Infrastructure," in Cloud Computing Technology and Science (CloudCom), Singapore, 2014.
20. T. H. Nguyen, D. F. Mario and Y. J. Antti, "A virtual machine placement algorithm for balanced resource utilization in cloud data centers," in Cloud Computing (CLOUD), Anchorage, 2014.
21. G. Yongqiang, G. Haibing, Q. Zhengwei, H. Yang and L. Liang, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," Journal of Computer and System Sciences, vol. 79, pp. 1230-1242, 2013.
22. M. Mayank and S. Anirudha, "On Theory of VM Placement: Anomalies in Existing Methodologies and Their Migration Using a Novel Vector Based Approach," in IEEE International Conference on Cloud Computing, 2011.
23. E. Barlaskar, Y. J. Singh and B. Isaac, "Energy-efficient virtual machine placement using enhanced firefly algorithm," Multiagent and Grid Systems, vol. 12, no. 3, pp. 167-198, 2016.
24. Kumar, S., Talwar, V., Kumar, V., Ranganathan, P., Schwan, K.: "Manage: Loosely Coupled Platform and Virtualization Management in Data Centers". In: Proceedings of the 6th international conference on Autonomic computing. pp. 127–136. ACM, Barcelona, Spain, June 2009.
25. Bobroff, N., Kochut, A., Beaty, K.: "Dynamic Placement of Virtual Machines for Managing SLA Violations". In: Proc of the 10th IFIP/IEEE International Symposium on Integrated Network Management. Munich, Germany, May 2007.
26. T. Wood et. al., "Black-box and gray-box strategies for virtual machine migration", proceedings of Symp. on Networked Systems Design and Implementation (NSDI), 2007
27. H. Zheng, L. Zhou, J. Wu, "Design and implementation of load balancing in web server cluster system", Journal of Nanjing University of Aeronautics & Astronautics, Vol. 38, No. 3, Jun. 2006
28. A. Singh, M. Korupolu, D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers", Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp 1-12, 2008.
29. E. Arzuaga, D. R. Kaeli, "Quantifying load imbalance on virtualized enterprise servers", Proceedings of WOSP/SIPEW'10, San Jose, California, USA, January 28-30, 2010
30. W. Tian, C. Jing, J. Hu, "Analysis of resource allocation and scheduling policies in cloud datacenter", Proceedings of the IEEE 3rd International Conference on Networks Security Wireless Communications and Trusted Computing, March 2011
31. T.Thiruvenkadam and Dr. V. Karthikeyani, "An approach to virtual machine placement problem in a datacenter environment based on overloaded resource", International Journal of Computer Science and Mobile Computing, Vol.3 No.6, pp. 837-842, 2014.
32. T.Thiruvenkadam and Dr.V.Karthikeyani, "A Comparative study of VM Placement Algorithms in Cloud Computing Environment", In proccedings of 07th SARC-IRF International Conference,August 2014,India.
33. Guo G., Ting-Iei H., Shuai G., " Genetic Simulated Annealing Algorithm for Task Scheduling based on Cloud Computing Environment", IEEE International Conference on Intelligent Computing and Integrated Systems (ICISS), Guilin, pp. 60-63, 2010.
34. Wilcox, D.,McNabb, A., Seppi, K.: "Solving virtual machine packing with a reordering grouping genetic algorithm". In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 362–369, 2011.
35. Tarun goyal & Aakanksha agrawal, "Host scheduling algorithm using genetic algorithm in cloud computing environment," International Journal of Research in Engineering & Technology (IJRET) Vol. 1, No 1, pp 7-12, June 2013.
36. M. Srinivas, and L. M. Patnaik, Fellow, IEEE. "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms". IEEE Transactions on Systems, Man and Cybernetics, Vol. 24, No. 4, April 1994.
37. P. Rohlfshagen and J. Bullinaria. "A Genetic Algorithm with Exon
38. Shu_ing Crossover for Hard Bin Packing Problems". Proceedings of
39. Genetic and Evolutionary Computation Conference, Vol.9,
40. pp-1365-1371, 2007.
41. T.Thiruvenkadam and Dr.V.Karthikeyani, "Multi Dimensional
42. HostLoad Aware and User Constraints Based Algorithm for Scheduling
43. Virtual Machines", International Journal of Advanced Computing
44. Technology (IJACT) ISSN: Vol. 7, No. 1, pg 56 – 66, 2015.
45. Pankajdeep Kaur and Anita Rani, " Virtual Machine Migration in Cloud Computing", International Journal of Grid Distribution Computing Vol. 8, No.5, (2015), pp.337-342
46. Debabrata Sarddar, Himadri Biswas and Priyajit Sen, "Safety as a Service (SFaaS) Model - The New Invention in Cloud computing to establish a Secure Logical Communication Channel between Data Owner and the Cloud Service Provider before Storing, Retrieving or Accessing any Data in the Cloud" International Journal of Grid and Distributed Computing, ISSN 2005-4262 Vol. 10, No. 10 (2017), pp.1-20

AUTHORS PROFILE

Himadri Biswas, Assistant Professor at the Department of Computer Science & Engineering, Modern Institute of Engineering & Technology, Bandel, Hooghly, West Bengal, INDIA. He also has 16 years of teaching experience and has served in several institutions. He is currently pursuing a Ph.D. degree from the University of Kalyani, Nadia, West Bengal, India. He's done his M. Tech in Computer Science & Engineering from WBUT in 2010 and his MCA from St. Xavier's College in Kolkata under IGNOU in 2004. His research interests include cloud computing, mobile computing.

Dr. Debabrata Sarddar, Assistant Professor at the Department of Computer Science and Engineering, University of Kalyani, Kalyani, Nadia, West Bengal, INDIA. He holds a Ph.D. degree at the University of Jadavpur. He completed his M.Tech in Computer Science & Engineering from DAVV, Indore in 2006 and his B.E. in Computer Science & Engineering from NIT, Durgapur in 2001. He published about 200 research papers in various journals, attended 50 conferences, wrote 10 book chapters and 3 books. His research interests include wireless and mobile systems, cloud computing and the wireless sensor network.