Table of Contents

Abstract

In this project, we define an architectural framework and principles for energy, performance and response efficient Cloud computing. For that, we investigate 2 problems in this paper: First, we investigate energy and computational efficient **VM allocation approach**. Second, we address performance, response efficient **Load Balancing Approach**. First, **VM allocation Approach** includes mapping of VMs to suitable Cloud resources (mainly host) and Virtual machine migration algorithms to dynamically consolidate virtual machines to minimum data center resources and there by putting off ideal host to either sleep or off mode to save energy consumption. In this report, we propose: architectural principles for energy-efficient management of Clouds and energy-efficient resource allocation policies and scheduling algorithms. The results demonstrate that Cloud computing model has immense potential as it offers significant cost savings and demonstrates high potential for the improvement of energy efficiency under dynamic workload scenarios. Second, **Load balancing** is the major concern in the cloud computing environment. In the present work, a novel VM-assign load balance algorithm is proposed which allocates the incoming requests to the all available virtual machines in an efficient manner. Further, the performance is compared with existing Active- VM load balance algorithm Simulation results demonstrate that the proposed algorithm distributes the load on all available virtual machines without under/over utilization.

1. Introduction

1.1 What is Cloud Computing?

Computing is being transformed to a model consisting of services that are commoditized, and in which users access services based on their requirements without regard to where the services are hosted or how they are delivered. Several computing paradigms have promised to deliver this utility computing vision and these include cluster computing, Grid computing, and more recently Cloud computing.

Cloud computing technology involves a large number of computers connected through a real-time communication such as the Internet. Cloud computing is basically distributed computing over a network, and means the ability to run a program or application on many connected computers at the same time. It also refers to network-based services, which appear to be provided by real server hardware and are in fact served up by virtual hardware, simulated by software running on one or more real machines. Such virtual servers do not physically exist and can therefore be moved around and scaled up (or down) on the fly without affecting the end user-rather like a cloud. Thus “Cloud” denotes the infrastructure through which businesses and users are able to access applications from anywhere in the world.

1.2 Need of High performance and efficient resource utilization:

Cloud offers significant benefit to IT companies by relieving them from the necessity in setting up basic hardware and software infrastructures. To fully realize the potential of Cloud computing, Cloud service providers have to ensure that they can be flexible in their service delivery to meet various consumer requirements without violating SLA (Service level Agreement) which can be ensured to some threshold if resources are utilized efficiently to ensure maximum possible system performance and minimum response time.

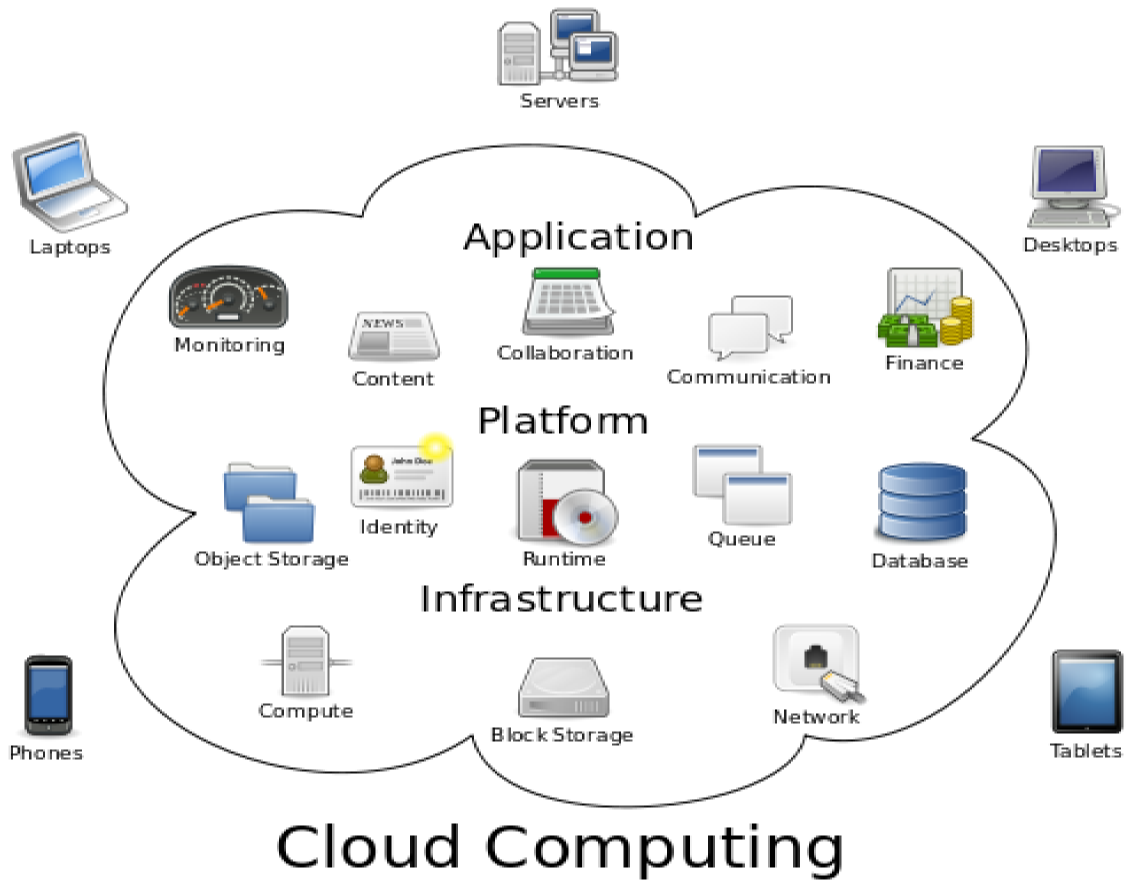
Load balancing is one of the key terms that affect the system performance dependent on the amount of work allotted to the system for a specific time period. Load balancing, as the name implies, is a technique that allows workloads to be distributed across multiple resources, to make effective resource utilization and to achieve better response time by handling a condition in which some of the nodes are over loaded while some others are under loaded. The aim of load balancing is to optimize utilization and throughput while reducing the response time. As on demand self service model, load arrives randomly or dynamically in cloud computing environment which causes some VM/servers to be heavily loaded while others idle or lightly loaded which in turn leads to poor performance and make user unsatisfied. So, if we distribute the load in proper way, it will improve system performance, throughput, response time etc. so as to meet user satisfaction. The important things to consider while developing such algorithm are: estimation of load, comparison of load, stability of different system, performance of system, interaction between the nodes, nature of work, node selection and many other ones.

1.3 Need of Green Cloud Computing:

Until recently, high performance has been the sole concern in data center deployments, and this demand has been fulfilled without paying much attention to energy consumption. However, an average data center consumes as much energy as 25,000 households [3]. As energy costs are increasing while availability dwindles, there is a need to shift the focus from optimizing data center resource management for pure performance to optimizing them for energy efficiency, while maintaining high service level performance. Lowering the energy usage of data centers is a challenging and complex issue because computing applications and data are growing so quickly that increasingly larger servers and disks are needed to process them fast enough within the required time period. Green Cloud Computing is envisioned to achieve not only the efficient processing and utilization of a computing infrastructure, but also to minimize energy consumption. To address this problem and drive Green Cloud computing, data center resources need to be managed in an energy-efficient manner. In particular, Cloud resources need to be allocated not only to satisfy Quality of Service (QoS) requirements specified by users via Service Level Agreements (SLAs), but also to reduce energy usage.

2. Characteristics of Cloud Computing

The characteristics of cloud computing include on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. On-demand self -service means that customers (usually organizations) can request and manage their own computing resources. Pooled resources means that customers draw from a pool of computing resources, usually in remote data centers. Services can be scaled and use of a service is measured and customers are billed accordingly.

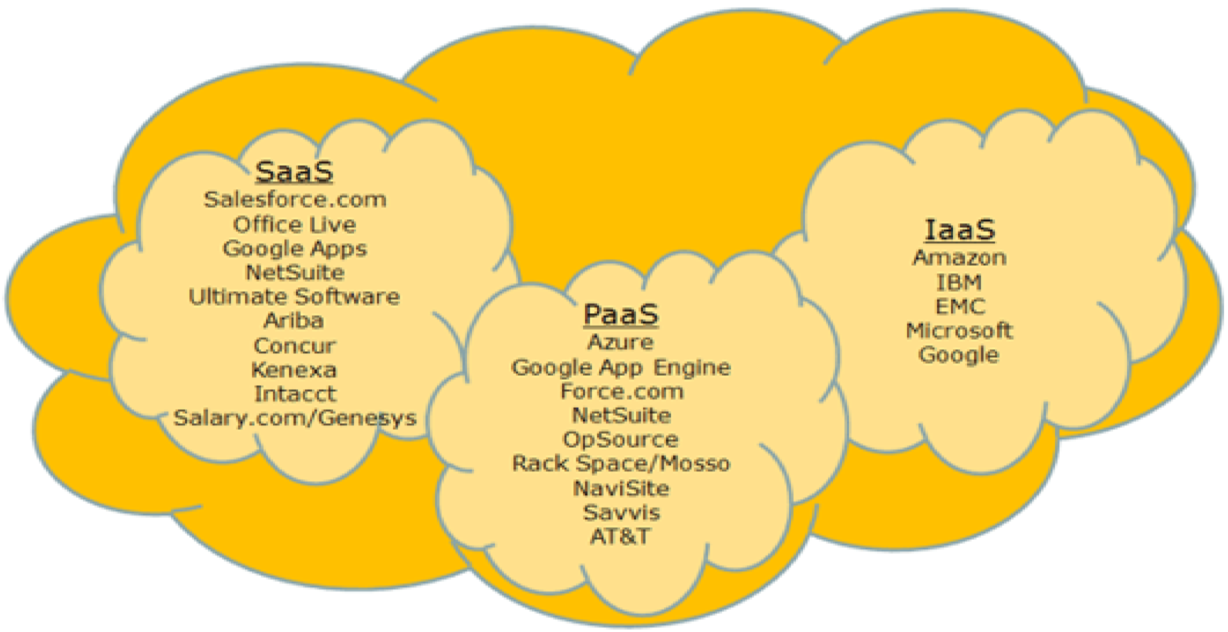


2.1 Cloud Services

Software as a service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the internet.

Platform as a service (PaaS) encapsulates a layer of software and provides it as service that can be used to build higher level services. There are at least two perspectives on PaaS depending on the perspective of the producer or consumer of the services.

Infrastructure as a service (IaaS) delivers basic storage and compute capabilities as standardized services over the network. Servers, storage systems, switches, routers and other systems are pooled and made available to handle workloads that range from application components to high performance computing applications.



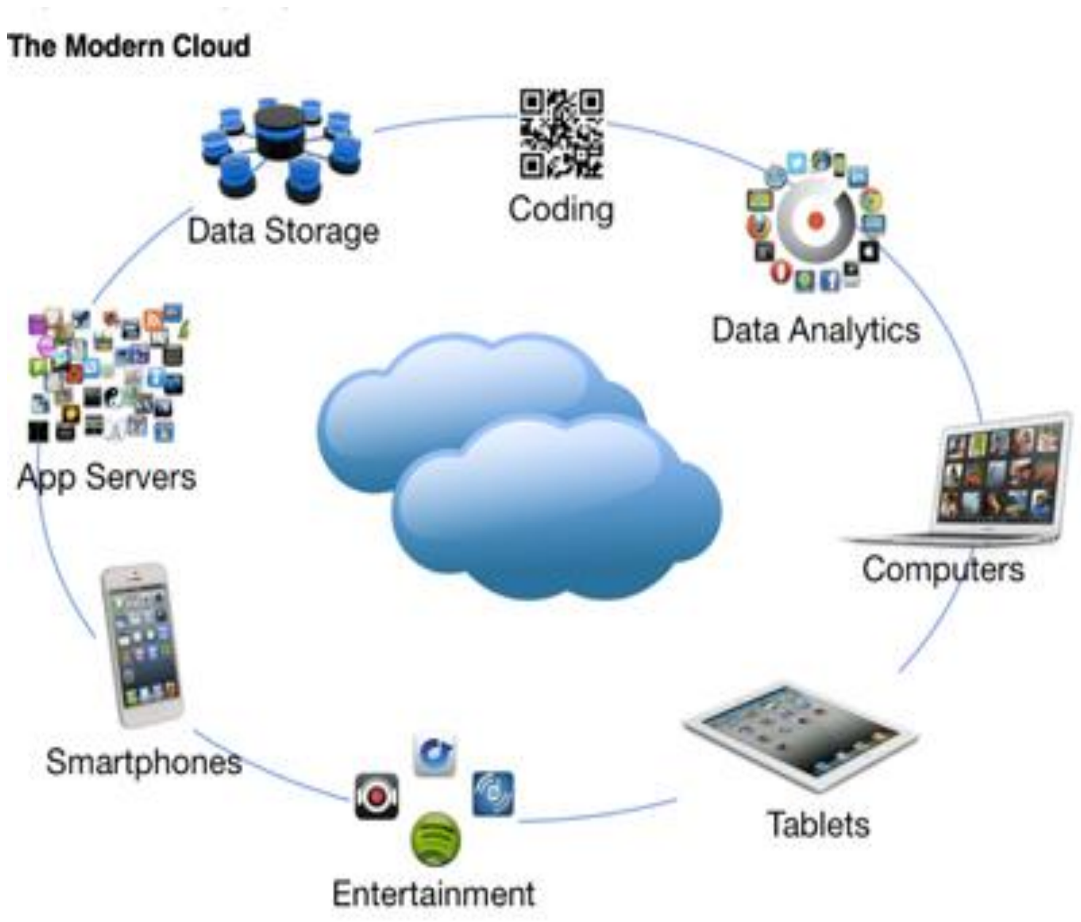
Cloud Computing Enablers - VMware, Adobe, Citrix, Akamai, Sun, Dell, HP, Red Hat

2.2 Cloud computing Models

Public Clouds: Public clouds are run by third parties and applications from different customers are likely to be mixed together on the cloud's servers, storage systems, and networks. Public clouds are most often hosted away from customer premises.

Private Clouds: Private clouds are built for the exclusive use of one client, providing the utmost control over data, security, and quality of service.

Hybrid Clouds: Hybrid clouds combine both public and private cloud models. They can help to provide on-demand, externally provisional scale.



2.3 The benefits and challenges of cloud computing

In recent years, cloud computing has emerged as an important solution offering enterprises a potentially cost effective model to ease their computing needs and accomplish business objectives. Wilson Law, a manager at member firm, Moore Stephens LLP Singapore, provides some key benefits below worth considering:

a) **Optimized server utilization** - as most enterprises typically underutilize their server computing resources, cloud computing will manage the server utilization to the optimum level.

b) **Cost saving** - IT infrastructure costs are almost always substantial and are treated as a capital expense (CAPEX). However, if the IT infrastructure usually becomes an operating expense (OPEX). In some countries, this results in a tax advantage regarding income taxes. Also, cloud computing cost saving can be realized via resource pooling.

c) **Dynamic scalability** - many enterprises include a reasonably large buffer from their average computing requirement, just to ensure that capacity is in place to satisfy peak demand. Cloud computing provides an extra processing buffer as needed at a low cost and without the capital investment or contingency fees to users.

For all the above benefits of cloud computing, it also incorporates some unique and notable technical or business risk as follows:

a) **Data location** - cloud computing technology allows cloud servers to reside anywhere, thus the enterprise may not know the physical location of the server used to store and process their data and applications. Although from the technology point of view, location is least relevant, this has become a critical issue for data governance requirements. It is essential to understand that many Cloud Service Providers (CSPs) can also specifically define where data is to be located.

b) **Commingled data** - application sharing and multi-tenancy of data is one of the characteristics associated with cloud computing. Although many CSPs have multi-tenant applications that are secure, scalable and customizable, security and privacy issues are still often concerns among enterprises. Data encryption is another control that can assist data confidentiality.

c) **Cloud security policy/procedures transparency**-some CSPs may have less transparency than others about their information security policy. The rationalization for such difference is the policies may be proprietary. As a result, it may create conflict with the enterprise's information compliance requirement. The enterprise needs to have detailed understanding of the service level agreements (SLAs)that stipulated the desired level of security provided by CSPs.

d) **Compliance requirements -** today's cloud computing services, can challenge various compliance audit requirements currently in place. Data location; cloud computing security policy transparency; and IAM, are all challenging issues in compliance auditing efforts. Examples of the compliance requirement including privacy and PII laws; Payment Card Industry (PCI) requirements; and financial reporting laws.

3. Related work

In this section, we briefly summarize the load balancing algorithms and VM allocation policies used in the cloud computing environment. The main focus is on the energy efficient utilization of the virtual machines and balancing the virtual machines with the incoming request.

1. Hemant S. Mahalle, Parag R. Kaveri and Vinay Chavan [3] have developed Active monitoring load balancer algorithm which maintains information about each VMs and the number of requests currently allocated to which VM. When a request to allocate a new VM arrives, it identifies the least loaded VM. If there are more than one, the first identified is selected. Active VM Load Balancer returns the VM id to the Data Center Controller the data Center Controller sends the request to the VM identified by that id. Data Center Controller notifies the Active VM Load Balancer of the new allocation.
2. Shridhar G. Domanal and G. Ram Mohana Reddy [4] have developed Modified Throttled algorithm which maintains an index table of virtual machines and also the state of VMs

similar to the Throttled algorithm [5]. There has been an attempt made to improve the response time and achieve efficient usage of available virtual machines. Proposed algorithm employs a method for selecting a VM for processing client's request where, VM at first index is initially selected depending upon the state of the VM. If the VM is available, it is assigned with the request and id of VM is returned to Data Center, else -1 is returned. When the next request arrives, the VM at index next to already assigned VM is chosen depending on the state of VM and follows the above step, unlikely of the Throttled algorithm, where the index table is parsed from the first index every time the Data Center queries Load Balancer for allocation of VM.

3. B.Wickremasinghe, R.N.Calheiros and Rajkumar Buyya have developed Throttled algorithm which is completely based on virtual machine. Here the client first requests the load balancer to check the right virtual machine which access that load easily and perform the operations which is given by the client [6]. In this algorithm the client first requests the load balancer to find a suitable Virtual Machine to perform the required operation In the present work we are considering Active-VM load balancer and proposed VM-assign algorithm for comparison. Our main focus is to distribute the load efficiently on the available virtual machines and ensuring that under or over utilization of the resources/ virtual machines will not occur in the cloud system.
4. Chase et al. [8] have considered the problem of energy-effi-cient management of homogeneous resources in Internet hosting centers. The main challenge is to determine the resource demand of each application at its current request load level and to allocate resources in the most efficient way. To deal with this problem the authors have applied an economic framework: services ‘‘bid’’ for resources in terms of volume and quality.
5. Nathuji and Schwan [10] have studied power management techniques in the context of virtualized data centers. Authors have introduced new power management technique called ‘‘soft resource scaling’’. The idea is to emulate hardware scaling by providing less resource time for aVM using theVirtualMachine Monitor’s (VMM) scheduling capability.
6. Kusic et al. [12] have defined the problem of power management in virtualized heterogeneous environments as a sequential optimization.The objective is to maximize the resource provider’s profit by minimizing both power consumption and SLA violation. Kalman filter is applied to estimate the number of future requests to predict the future state of the system and perform necessary reallocations.
7. Cardosa et al. [14] have proposed an approach for the problem of power-efficient allocation of VMs in virtualized hetero-geneous computing environments. They have leveraged the min, max and shares parameters of VMM, which represent minimum, maximum and proportion of the CPU allocated to VMs sharing the same resource.
8. Verma et al. [15] have formulated the problem of power-aware dynamic placement of applications in virtualized heterogeneous systems as continuous optimization: at each time frame the placement of VMs is optimized to minimize power consumption and

maximize performance. The authors have applied a heuristic for the bin packing problem with variable bin sizes and costs. The proposed algorithms, on the contrary to our approach, do not handle strict SLA requirements: SLAs can be violated due to variability of the workload.

4. Discussed Approach

Green Cloud computing is envisioned to achieve not only the efficient processing and utilization of a computing infrastructure, but also to minimize energy consumption.

Energy efficient heuristics approach for VM allocation and placement

To minimize energy consumption, we propose efficient heuristics for dynamic adaption of VM allocation at run-time according to the current utilization of resources applying live migration, switching idle nodes to the sleep mode, and thus minimizing energy consumption. The algorithms do not depend on a particular type of workload and do not require any knowledge about applications running in VMs. The optimization of the current VM allocation is carried out in two steps:

1. We select VMs that need to be migrated (Minimum Migration policy)
2. Selected VMs are placed on the hosts using the improved MBFD algorithm.

4.1 The Minimization of Migrations policy

The Minimization of Migrations (MM) policy selects the minimum number of VMs needed to migrate from a host to lower the CPU utilization below the upper utilization threshold if the upper threshold is violated.

Data Structure Used:

Binary search search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index, or position, is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

A binary search halves the number of items to check with each iteration, so locating an item (or determining its absence) takes logarithmic time. A binary search is a dichotomic divide and conquer search algorithm.

Worst case performance	O(log n)
Best case performance	O(1)
Average case performance	O(log n)

Worst case space complexity	O(1)

Algorithm Used:

The pseudo-code for the Minimization of migration algorithm for the over-utilization case is presented in here. The algorithm sorts the list of VMs in the decreasing order of the CPU utilization. Then, it repeatedly looks through the list of VMs and finds a VM that is the best to migrate from the host. The best VM is the one that satisfies two conditions. First, the VM should have the utilization higher than the difference between the host's overall utilization and the upper utilization threshold. Second, if the VM is migrated from the host, the difference between the upper threshold and the new utilization is the minimum across the values provided by all the VMs. If there is no such a VM, the algorithm selects the VM with the highest utilization, removes it from the list of VMs, and proceeds to a new iteration. The algorithm stops when the new utilization of the host is below the upper utilization threshold. The complexity of the algorithm is proportional to the product of the number of over-utilized hosts and the logarithm of number of VMs allocated to these hosts. As we will do binary search instead of linear in virtual machine list as the list is sorted in decreasing order in log (vmList).

hostList – List of all hosts in cloud

migrationList – List of virtual machines to be migrated from host

THRESH_UP – Maximum utilization that any host can beer

THRESH_LOW – Minimum utilization that any host can beer

sortDecreasingUtilization() – Sort the list in decreasing order in terms of utilization

getVmList() – Returns the list of virtual machines on a particular host

hUtil – Temporary variable to store the host utilization

Algorithm : Minimization of Migration (MM)

Input : hostList **Output** : migrationList

```
foreach h in hostList do
    vmList = h.getVmList()
    vmList.sortDecreasingUtilization()
    hUtil = h.getUtil()
    while hUtil > THRESH_UP do
```

```
        bestFitVm = binarySearch (vmList, vm.getUtil()  >  hUtil – THRESH_UP)
    hUtil = hUtil – bestFitVm.getutil()
    migrationList.add(bestFitVm)
    vmList.remove(bestFitVm)
    if hUtil < THRESH_LOW then
        migrationList.add(h.getVmList())
        vmList.remove(h.getVmList())
return migrationList
```

Here, binarySearch (vmList, vm.getUtil() > hUtil – THRESH_UP) defines binary search on virtual machines utilization list. In this, we will do binary search for the element which is greater than difference of hUtil and THRESH_UP in vmList which is sorted in decreasing order.

4.2 VM Placement

The problem of VM allocation can be divided in two: the first part is the admission of new requests for VM provisioning and placing the VMs on hosts, whereas the second part is the optimization of the current VM allocation. The first part can be seen as a bin packing problem with variable bin sizes and prices. To solve it we apply a modification of the Best Fit Decreasing91 bins (where *OPT* is the number of bins given by the optimal solution) .In our modification, the Modified Best Fit Decreasing (MBFD) algorithms, we sort all VMs in decreasing order of their current CPU utilizations, and allocate each VM to a host that provides the least increase of power consumption due to this allocation.

Data Structure-> Heap:

For choosing most power efficient node, we keep resources with their utilization and maximum capacity in minimum heap. In optimizing allocation of VM to the hosts when we have set of hosts and set of virtual machines with the decreasing order of their utilization, then it becomes smart to use min heap for host to find minimum current utilizing host that linearly searching so that when a VM is allocated to it ,it gives rises to minimum increase of power consumption due to allocation of new VM.

Min heap utilized to store set of hosts utilizes properties of heap for efficiently accessing minimum element in O(1).

Efficiency of heaps

Assume the heap has N nodes. Then the heap has log2(N+1) levels.

- Since the insert swaps at most once per level, the order of complexity of insert is O(log N)

- Since the remove swaps at most once per level, theorder of complexity of remove is also O(log N)
- Accessing top element O(1)

The pseudo-code for the algorithm is presented in Algorithm 1. The complexity of the allocation part of the algorithm is $n \cdot log(m)$, where n is the number of VMs that have to be allocated and m is the number of hosts.

Modified Efficient Algorithm for Virtual Memory Allocation

```
Input:
1. hostlist=List of available hosts in the cloud.
2. vmlist=List of available virtual machines to be allocated hosts.
3. vm = the utilization of virtual machine for any host.
4. host = power utilization of the hosts in hostlist.
5. sortDecreasingUtilization()= Sort entity(vm or host) in decreasing order of
   utilization.
6. createMinHeap()=Creates heap of entity(vm or host) as per their utilization.
Algorithm
7. allocatedHost=Allocated for Vm
8. minheapTop()= Returns root value of heap.
    A. Sort VM in vmlist in decreasing order of utilization
    Vmlist. sortDecreasingUtilization();
    B. Create min Heap of hosts according to their utilization.
    host.createMinHeap(hostlist);
    C. Foreach vm in vmlist Do
        { allocatedHost=NULL

          MaxhostUtil=host.minheapTop();
        Temp=host.minheapTop();
      If(MaxhostUtil-Temp>=vm)//host has enough resource for vm
        {   allocatedHost=host;// allocated host for Vm is host
           host.hUtil=Temp+vm;
           Heapify(hostlist,1);
        }

        }
        Return allocatedHost
```

4.3 Algorithm: VM-Assign Load Balancer

In the present work we are considering Active-VM load balancer and proposed VM-assign algorithm for comparison. Our main focus is to distribute the load efficiently on the available virtual machines and ensuring that under or over utilization of the resources/ virtual machines will not occur in the cloud system.

Input: No of incoming jobs X1, X2......... Xn

Available: VM Y1, Y2 ………...... Yn

Output: All incoming jobs X1, X2........ Xn are allocated least loaded virtual machine among the available Y1, Y2 …….. Yn

1: Initially all the VM's have 0 allocations.

2: VM-assign load balancer maintains the index / assign table of VMs which has no. of requests currently allocated to each VM.

3: When requests arrive at the data center it passes to the load balancer.

4: Index table is parsed and least loaded VM is selected by serially increasing the previously index and taking modulo for value higher than number of virtual machines.

Case I: if found

Least loaded VM is chosen

5: VM-assign load balancer returns the VM id to the data center.

6: Request is assigned to the VM. Data center notifies the VM-assign load balancer about the allocation.

7: VM-assign load balancer updates the requests hold by each VM.

8: When the VM finishes the processing the request, data center receives the response.

9: data center notifies the VM-assign load balancer for the VM de-allocation and VM-assign load balancer updates the table.

10: Repeat from step 2 for the next request.

CIRCULAR VM-ASSIGN LOAD BALANCE ALGORITHM

This algorithm focuses mainly on finding out the least loaded virtual machine and how incoming jobs are allocated intelligently. The basic methodology of the proposed VM - assign algorithm is given in the following Fig. I

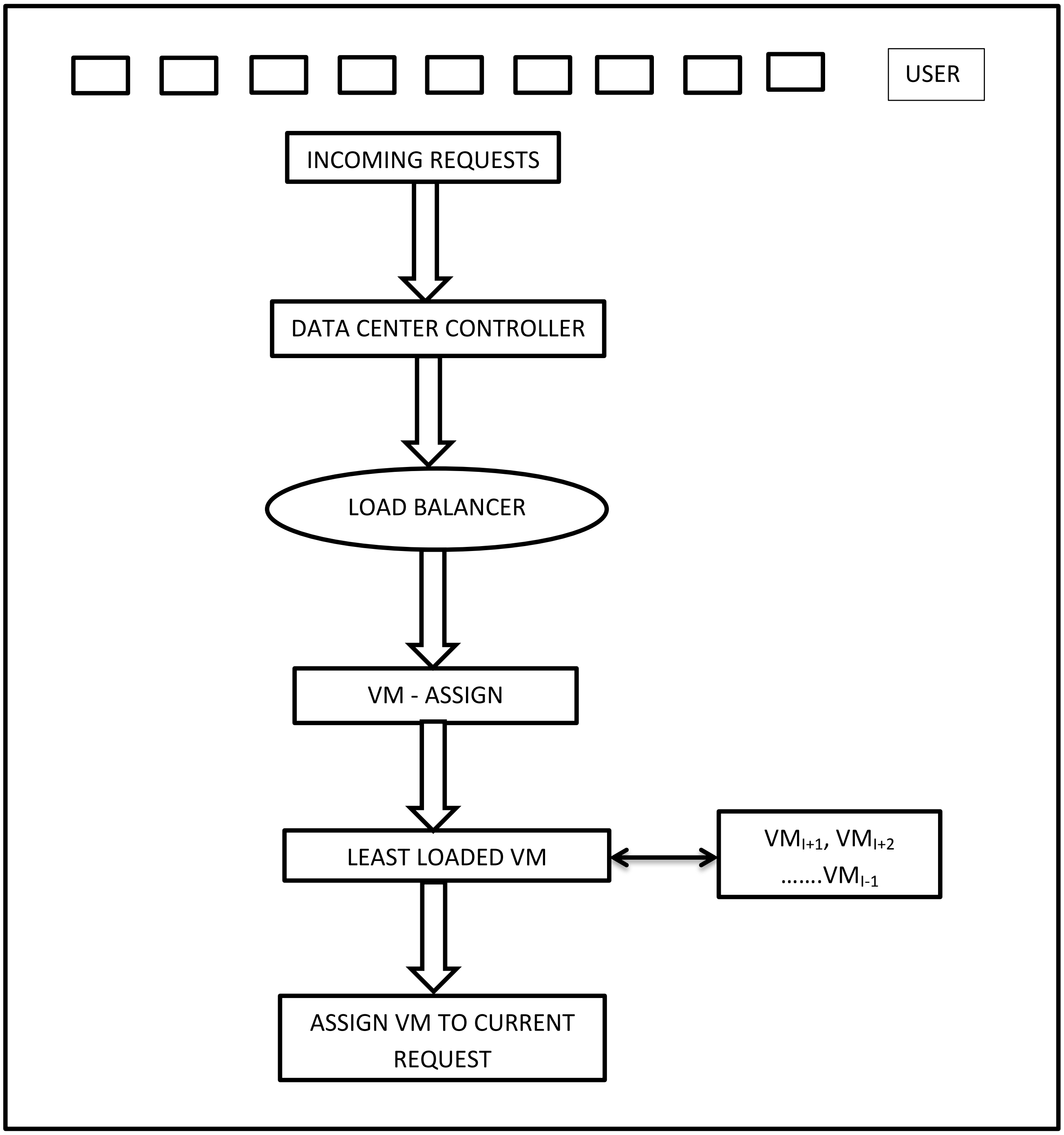


Fig: Flow of the Proposed VM-assign Algorithm

5. Simulation and result

5.1 For Proposed VM Placement Policy

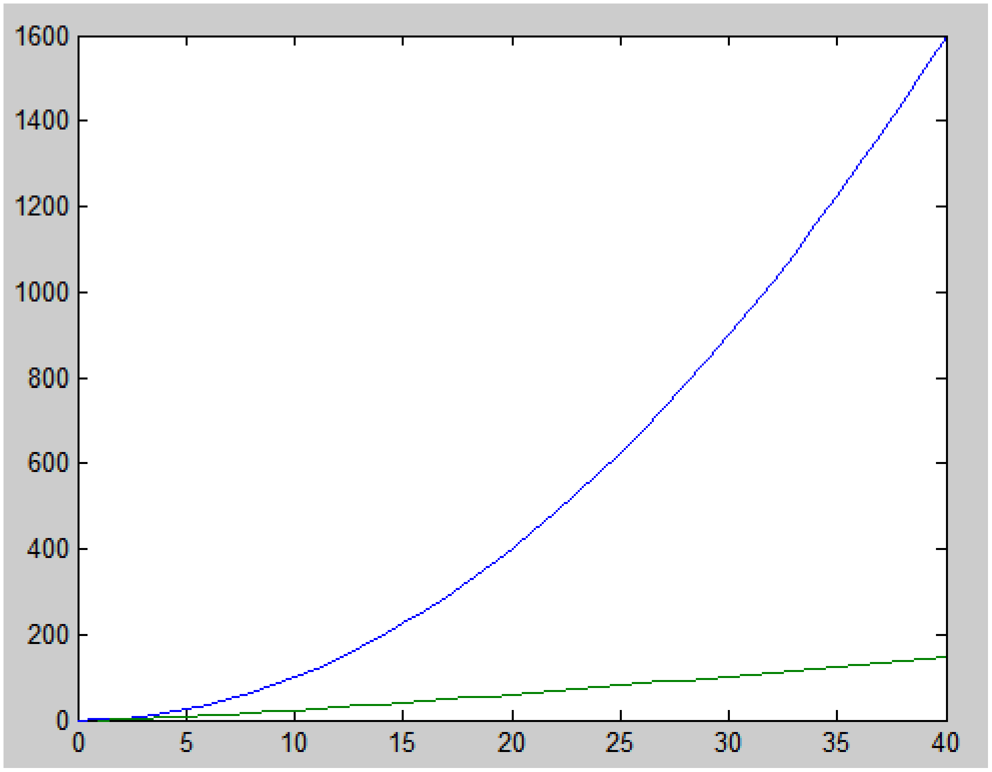
a. Complexity Analysis:

MBFD vs Proposed Algorithm.

Current Complexity: O(n.log(m))

Previous Complexity: O(n.m)

n=Number of VM requested.
m=Number of Host available



So, there is an improvement in proposed algorithm which works with time complexity of O(nlog(m)) as compared to given MBFD algorithm with time complexity of O(n.m)

b. SLA violations: Results related to SLA violations are very similar to given MBFD algorithm but there is good improvement in violation as compared to random allocation policy. In random allocation policy, there is probability of allocating a VM to host running with 100% utilization even when there is availability of various hosts with much lower utilization. So, in this case, VM will not get required resources leading to SLA violation.

Comparing results with random allocation policy.

Algorithm	Host(number)	VM(number)	Average SLA Violation
Random Access	5	10	10%
Proposed Algo (Improved MBFD)	5	10	0%
Random Access	12	30	17.56%
Proposed Algo (Improved MBFD)	12	30	10.25%

c. Performance improvement due to load balancing at provider (server) end.

Choosing a host with minimum utilization among all available hosts for allocation of VM results into a kind of load balancing which is far better than random allocation policy. In a random allocation policy, there is chance of underutilization and over utilization of resources which leads to performance degradation, less through put and high response time. In our algorithm, we have improved all these factors while comparing to MBFD given algorithm, result are very much similar.

5.2 For Proposed Minimum Migration Policy

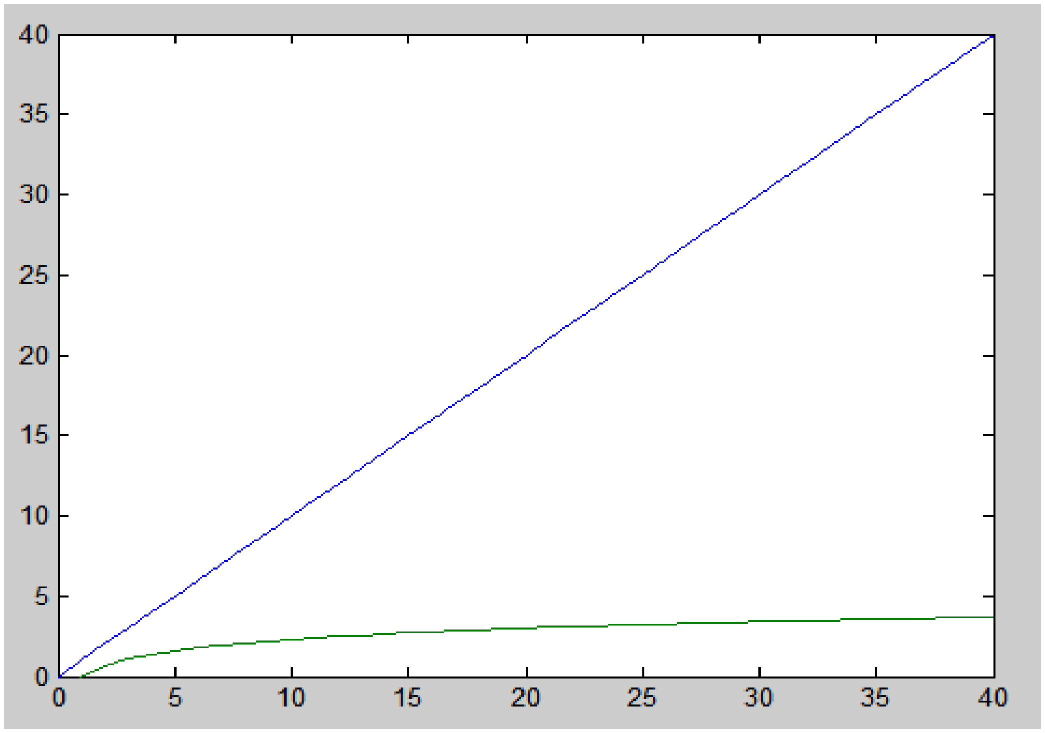
5.2.1 Minimized Migration Benefits:

- Energy Benefits
- Load Benefits
- Online Maintenance
- Increased lifetime of hosts
- Reliability of hosts
- Preventing Performance degradation
- Fulfilling the criteria of QoS
- Higher Performance
- Improved manageability

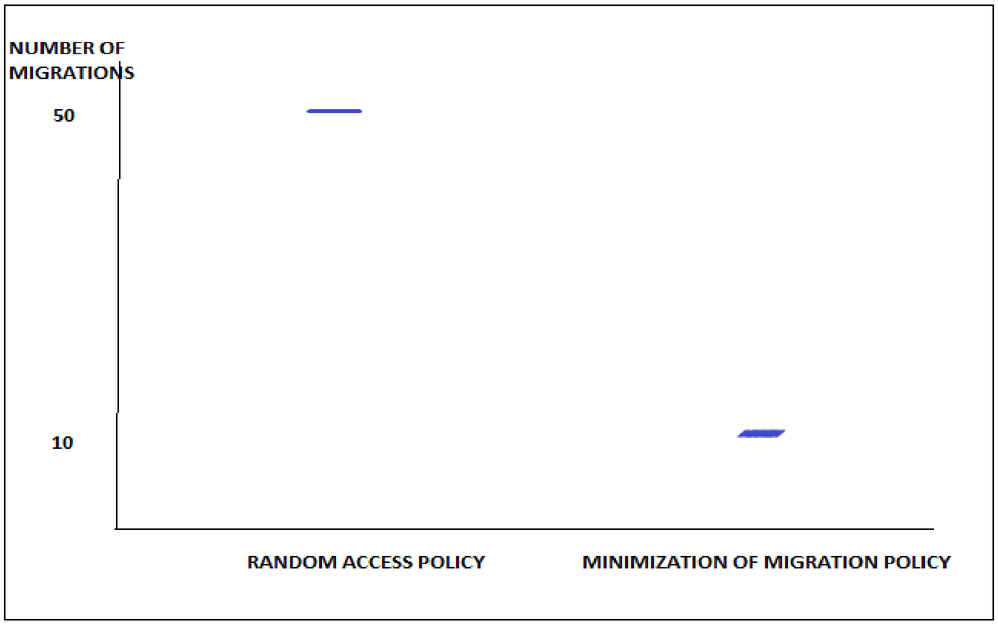
5.2.2 Complexity Analysis:

Current Complexity: O(log(n))

Previous Complexity: O(n)



5.2.3 Overhead in migration: In this mechanism, Minimum number of migrations will be required as compared to random access policy. In random access policy, randomly 1 virtual machine will be selected from top and it will be migrated. In minimum migration scheme, appropriate set of virtual machines is selected for migration with the help of which minimum virtual machines are migrated and we can utilize the host to its maximum value.



5.2.4 SLA Violation improvement: Results related to SLA violations are very similar to given MM algorithm but there is good improvement in violation as compared to random allocation policy. In random allocation policy, there is probability of allocating a VM to host running with 100% utilization even when there is availability of various hosts with so much lower utilization. So, In this case, VM will not get required resources leading to SLA violation.

5.3 For VM Load Balancing

Proposed algorithm will not allow the VM which was allocated in its previous steps. But this is not the same case with Active load balancing algorithm in which it assigns the least loaded VM depending on the current load and in VM – assign Load Balancer the load can be assigned again and again to same virtual machine. So with this in Active load balancer and VM-assign load balancer algorithm few VMs are overloaded with many requests and remaining VMs will handle only few requests with this under utilization of the VMs takes place. This results in imbalance of the load on the VMs. But if we use the proposed Circular VM-assign load balance algorithm all the VMs are utilized completely and properly. Our algorithm proves that there is no under utilization of the resources in the cloud. The algorithm is tested for initially with three VM. In this case our proposed algorithm balances the load on all available VMs in an efficient way. Hence we can say that our algorithm will overcome the under lover utilization of resources usage problem. The following Table gives the information about how many times each VM has been used efficiently.

SI. No.	Active Load Balancer(FIFO)	VM – assign Load Balancer	FIFO VM – assign Load Balancer
VM0	68	36	26
VM1	5	34	24
VM2	2	5	25

From the Table, we can observe that using Circular VM-assign load balancer distributes the incoming requests to all VM's in an intelligent way compared to Active load balancer and VM-assign load balancer. In the proposed algorithm we can see, utilization of the resources is neither underutilized nor over utilized but where as in Active load balancer VM 0 is over utilized and VM 2 is never utilized and other VM's are underutilized and in VM-assign load balancer, although we avoid assignment of load to previous Virtual Machine but if there is case that load is removed from virtual machine till the new load arrives, then allocation will be localized to few virtual machines.

6. Conclusion

Our work plays a significant role in the reduction of data center energy consumption costs, performance costs and response time and thus helps to develop a strong and competitive Cloud computing industry.

For VM allocation algorithm, Complexity has been reduced a lot using min-heap technique, efficient computation leading to less overload to central server to decide placement of virtual machines on each host.

For VM migration algorithm, Complexity has been reduced using binary search method which has led to energy saving, load balancing, online maintenance, increase of hosts, reliability of hosts, preventing performance of degradation, fulfilling the criteria of QoS, higher performance and improved manageability. This approach leads to a substantial reduction of energy consumption in Cloud data centers in comparison to static resource allocation techniques.

For load balancing, an efficient algorithm is designed which manages the load at the server by considering the current status of the all available VMs for assigning the incoming requests intelligently. The VM-assign load balancer mainly focuses on the efficient utilization of the resources/VMs. We proved that our proposed algorithm optimally distributes the load and hence under / over utilization (VMs) situations will not arise. When compared to existing Active-VM load balance algorithm, the load was not properly distributed on the VMs. It proves that initial VMs are over utilized and later VMs are underutilized. Our proposed algorithm solves the problem of inefficient utilization of the VMs / resources compared to existing algorithm.

7. Future Scope

As a future scope the proposed algorithm for load balancer can still be improved by taking some more dynamic situations of the incoming requests and how the algorithm responses if we mix both static and dynamic loads. By considering the proper optimization of network interface, RAM, disk storage and virtual network topologies, we can further optimize the system.

8. Bibliography

1. Anton Beloglazov a, Jemal Abawajyb, Rajkumar Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing, 2011.
2. R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616.
3. R. Buyya, A. Beloglazov, J. Abawajy, Energy-efficient management of datacenter resources for cloud computing: a vision, architectural elements, and open challenges, in:

Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2010, Las Vegas, USA, 2010.

4. Pinheiro, R. Bianchini, E.V. Carrera, T. Heath, Load balancing and unbalancing for power and performancee in cluster-based systems, in: Proceedings of the Workshop on Compilers and Operating Systems for Low Power, 2001, pp. 182–195.
5. M. Gupta, S. Singh, Greening of the internet, in: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM 2003, New York, NY, USA, 2003, pp. 19–26.
6. R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, X. Zhu, No ‘‘power’’ struggles: coordinated multi-level power management for the data center, SIGARCH Computer Architecture News 36 (1) (2008) 48–59.
7. Binary Search algorithm http://en.wikipedia.org/wiki/Binary_search_algorithm

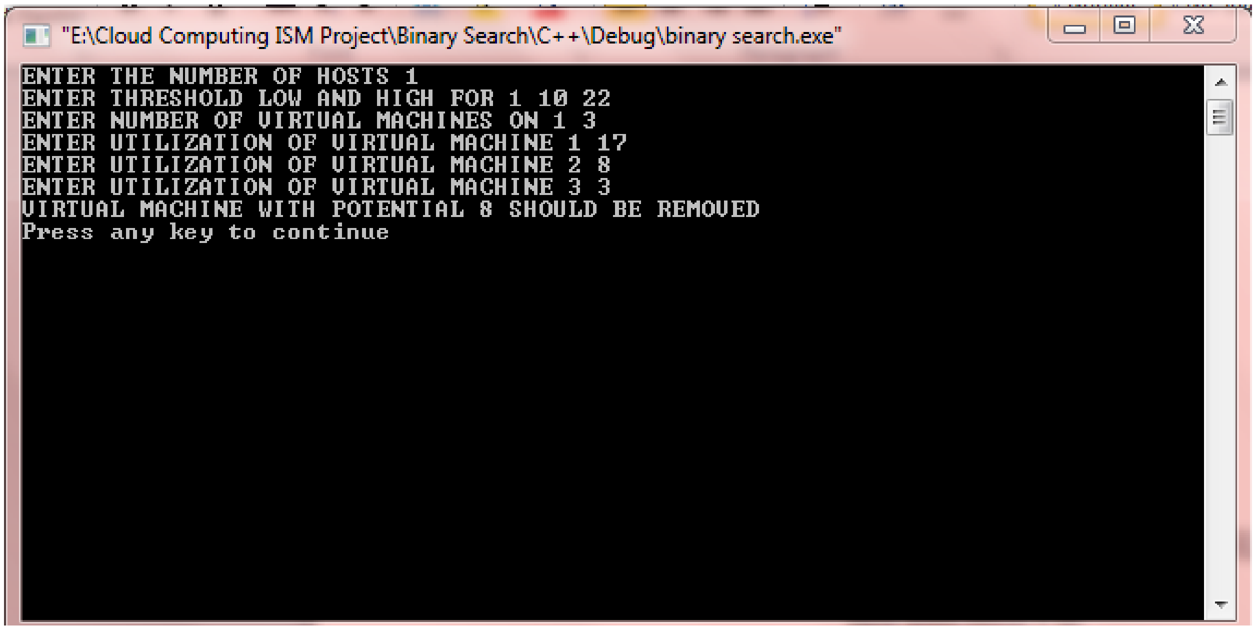
9. Appendix

9.1 Code for VM Migration

```
#include<iostream>

#include<vector>

using namespace std;

class hosts{

public:

int threshUp, threshLow, vm, *vmList;

   int sum(int arr[], int start, int end){

         int i, s = 0;

         for(i = start; i <= end; i++){

              s += arr[i];

}

return s;

   }

int binarySearch(int a[], int start, int end, int search){

         if(a[end] > search){
            printf("VIRTUAL MACHINE WITH POTENTIAL %d SHOULD
BE REMOVED\n", a[end]);

            return end;

        }

        if(a[start] < search){

            printf("VIRTUAL MACHINE WITH POTENTIAL %d SHOULD
BE REMOVED\n", a[start]);

            return -1;

        }

        while(start < end){

            if(a[(start + end)/2] > search && a[(start +
end)/2 + 1] <= search){

                printf("VIRTUAL MACHINE WITH POTENTIAL %d
SHOULD BE REMOVED\n", a[(start + end)/2]);

                return (start + end)/2;

            }else if(a[(start + end)/2] > search){

                start = (start + end)/2;

            }else{

                end = (start + end)/2;

            }

        }

        printf("VIRTUAL MACHINE WITH POTENTIAL %d SHOULD BE
REMOVED\n", a[(start + end)/2]);

        return (start + end)/2;

   }

};

int main(){
   int n, i, j;
    printf("ENTER THE NUMBER OF HOSTS ");
    scanf("%d", &n);
   hosts *h = (hosts *)malloc(sizeof(hosts)*n);
   for(i = 0; i < n; i++){
        printf("ENTER THRESHOLD LOW AND HIGH FOR %d ", i + 1);
        scanf("%d %d", &h[i].threshLow, &h[i].threshUp);
        printf("ENTER NUMBER OF VIRTUAL MACHINES ON %d ", i +
1);          scanf("%d", &h[i].vm);
        h[i].vmList = (int *)malloc(sizeof(int)*(h[i].vm));
        for(j = 0; j < h[i].vm; j++){
        printf("ENTER UTILIZATION OF VIRTUAL MACHINE %d ", j +
1);
        scanf("%d", &h[i].vmList[j]);
        }
   }
   int hUtil, start, end, search, index;
   vector<int> migrationList;
   for(i = 0; i < n; i++){
        start = 0;
        end = h[i].vm - 1;
        while((hUtil = h[i].sum(h[i].vmList, start, end)) >
h[i].threshUp){
             search = hUtil - h[i].threshUp;
             if((index = h[i].binarySearch(h[i].vmList, start,
end, search)) != -1){
                  migrationList.push_back(index);
```

```
                    h[i].vmList[index] = 0;

                    start = start + 1;

                }else{

                    migrationList.push_back(h[i].vmList[start]);

                    h[i].vmList[start] = 0;

                    start = start + 1;

                }

            }

            if(hUtil < h[i].threshLow){

                for(j = 0; j < h[i].vm; j++){

                    migrationList.push_back(h[i].vmList[j]);

                    h[i].vmList[j] = 0;

                }        }    }

    return 0;

}
```

OUTPUT:



"E:\Cloud Computing ISM Project\Binary Search\C++\Debug\binary search.exe"

```
ENTER THE NUMBER OF HOSTS 1
ENTER THRESHOLD LOW AND HIGH FOR 1 10 22
ENTER NUMBER OF VIRTUAL MACHINES ON 1 3
ENTER UTILIZATION OF VIRTUAL MACHINE 1 17
ENTER UTILIZATION OF VIRTUAL MACHINE 2 8
ENTER UTILIZATION OF VIRTUAL MACHINE 3 3
VIRTUAL MACHINE WITH POTENTIAL 8 SHOULD BE REMOVED
Press any key to continue
```

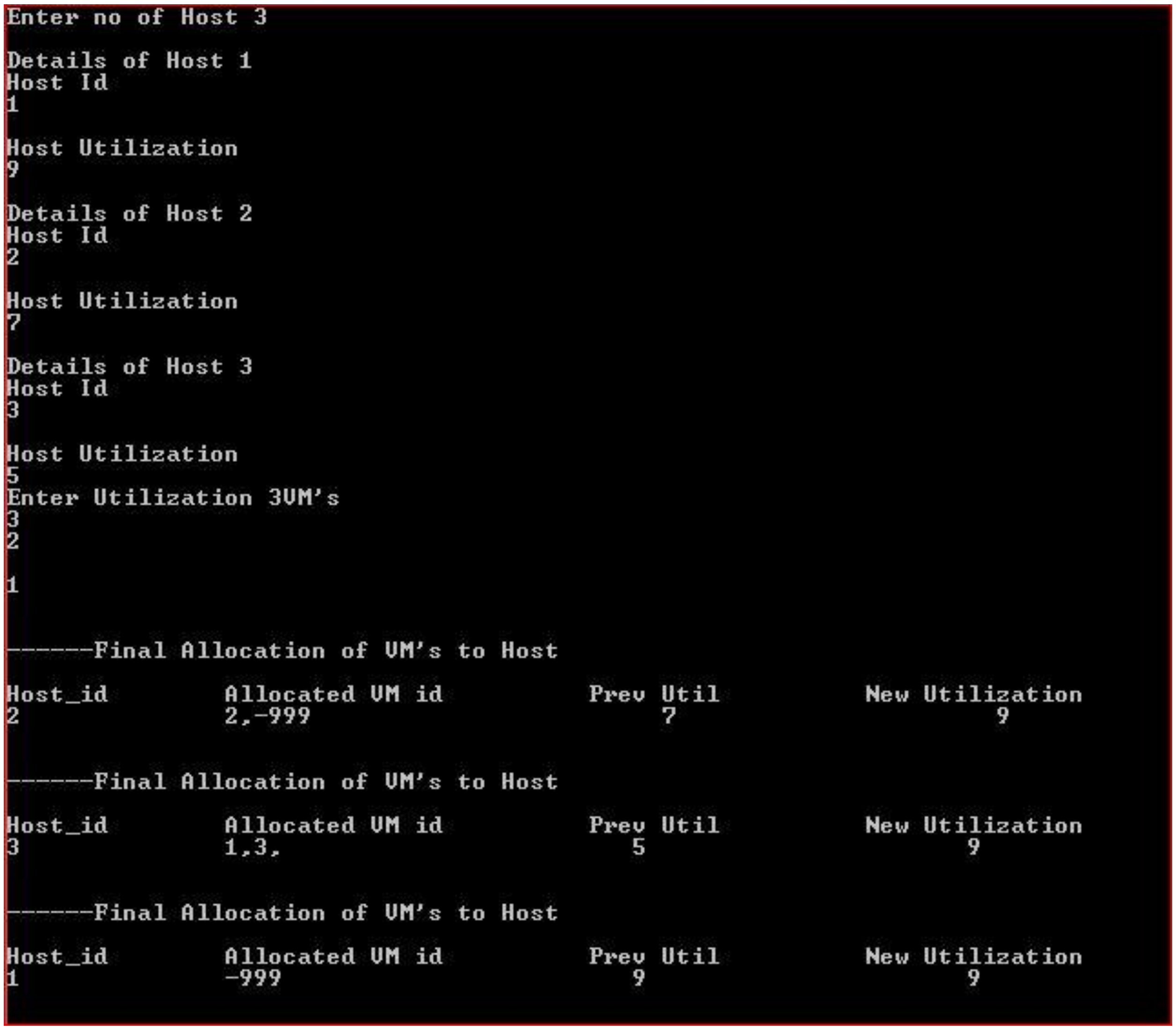
9.2 Code for VM Allocation

```
#include<iostream>

#include<conio.h>

#include<stdlib.h>

#define VM_size 3

#define size 2


using namespace std;

int allochost[VM_size+1];

int newallochost[VM_size+1];

class Host

{

int counter;

int util;

int id;

int allocated_vm[size+1];

int newutil;

public:

       Host(){}

       void get_input(void);

       void show_output(int);

       friend void build_minheap(Host hostob[],int nhost);

       friend void min_heapify(Host hostob[],int i,int nhost);

       friend void AllocateVM(Host hostob[],int nhost,int 
Vm_util[],int nvm,int allochost_id[]);
```

```
        };
void Host ::get_input(void)
{int i;
counter=0;
for(i=1;i<=size;i++)
allocated_vm[i]=-999;
cout<<"Host Id"<<"\n";
cin>>id;
cout<<"\n"<<"Host Utilization"<<"\n";
cin>>util;
newutil=util;
}
void Host ::show_output(int i)
{       cout<<"\n"<<"\n"<<"------Final Allocation of VM's to
Host"<<"\n"<<"\n";
     cout<<"Host_id"<<"        "<<"Allocated VM id"<<"
"<<"Prev Util"<<"          "<<"New Utilization"<<"\n";
     cout <<id <<"              ";
  int j=1;
  while(allocated_vm[j]!=-999 &&j<=size)
  {  cout <<allocated_vm[j];
  cout<<",";
  j++;
  }  if(j<=size)
  cout <<allocated_vm[j];
  cout<<"                       "<<util<<"
"<<newutil<<"\n";

}

void build_minheap(Host hostob[],int nhost)

{int i;

  for(i = nhost/2; i >= 1; i--)

    {        min_heapify(hostob,i,nhost);    }

}

void min_heapify(Host hostob[],int i,int nhost)

{ Host tempob;

    tempob=hostob[i];

   int temp,temp_id;

    int j;

    temp = hostob[i].newutil;

    temp_id=hostob[i].id;

    j = 2 * i;

    while (j <= nhost)

    {        if (j < nhost && hostob[j+1].newutil <
hostob[j].newutil)

            j = j + 1;

        if (temp < hostob[j].newutil)

            break;

        else if (temp >= hostob[j].newutil)

        {

            hostob[j/2]=hostob[j];

            j = 2 * j;
        }
    }
    hostob[j/2]=tempob;
    return;
}
void AllocateVM(Host hostob[],int nhost,int Vm_util[],int
nvm,int allochost_id[])
{int max=100;
     int i,j,k;
     int min;
     for(i=1;i<=nvm;i++)
     {allochost[i]=0;
     newallochost[i]=0;
     }
   for(i=1;i<=nvm;i++)
   {   min=hostob[1].newutil;
        if(min+Vm_util[i]<=max)
        {
        allochost[i]=min;
        newallochost[i]=min+Vm_util[i];
        allochost_id[i]=hostob[1].id;
        hostob[1].allocated_vm[++(hostob[1].counter)]=i;
        hostob[1].newutil=min+Vm_util[i];
        min_heapify(hostob,1,nhost);
        }    }
}
```

```
int main(){
    int nhost,i;
    cout <<"Enter no of Host";
    cin>>nhost;
 Host hostob[nhost+1];
 for(i=1;i<=nhost;i++){
 cout<<"\n"<<"Details of Host "<<i<<"\n";
 hostob[i].get_input();
  }
int Vm_util[VM_size+1];
cout<<"Enter Utilization "<<VM_size<<"VM's"<<"\n";
int j;
for(j=1;j<=VM_size;j++)
cin>>Vm_util[j];
//Creating Heap
build_minheap(hostob,nhost);

int allochost_id[nhost+1];
for(i=1;i<=nhost;i++)
allochost_id[i]=-99;
AllocateVM(hostob,nhost,Vm_util,VM_size,allochost_id);
//output

for(i=1;i<=nhost;i++){
    hostob[i].show_output(i);
}


getch();

return 0;

}
```

OUTPUT:



```
Enter no of Host 3

Details of Host 1
Host Id
1

Host Utilization
9

Details of Host 2
Host Id
2

Host Utilization
7

Details of Host 3
Host Id
3

Host Utilization
5
Enter Utilization 3VM's
3
2

1


------Final Allocation of VM's to Host

Host_id        Allocated VM id          Prev Util          New Utilization
2              2,-999                        7                          9

------Final Allocation of VM's to Host

Host_id        Allocated VM id          Prev Util          New Utilization
3              1,3,                        5                          9

------Final Allocation of VM's to Host

Host_id        Allocated VM id          Prev Util          New Utilization
1              -999                        9                          9
```