# Incorporating Migration Control in VM Selection Strategies to Enhance Performance

Mohammad Alaul Haque Monil, Romasa Qasim, Rashedur M Rahman
Department of Electrical and Computer Engineering (ECE)
North South University Dhaka, Bangladesh
{monil01, its.romi@gmail.com}, rashedur@northsouth.edu

***ABSTRACT:*** *In this work we have designed three algorithms, by doing amalgamation of heuristic method and migration control. Virtual Machine (VM) selection is the immediate step after the host overload is detected. Now decision needs to be made to choice the VM(s) to migrate. VM selection is an important task and the energy consumption depends on the selection. Three VM selection strategies have been proposed in literature and been incorporated in CloudSim, a simulation framework to simulate Cloud architecture, resource provisioning etc. Migration control could influence the VM selection mechanism as steady and resource consuming VMs not to be selected. Reason behind is that, a steady VM which is continuously consuming recourses has high probability of consuming resource in the same manner in future. So if we migrate such VM it will also overload the next datacenter in which we are migrating it to. In this proposed model, CPU usage has been considered to identify a VM to be steady or not. The VM selection mechanisms are modified and equations are also modified to adopt migration control in the policy and algorithms also developed. We have used Cloudsim toolkit to simulate our experiment and to evaluate the performance of the proposed algorithm, we have used real-world work load traces of Planet lab VMs. From the comparison with existing energy aware VM consolidation methods, it is found that performance of our algorithm is better than other proposed heuristic approaches.*

**Keywords:** Cloud, Datacenter, Dynamic Virtual Machine Consolidation, Cloudsim Toolkit, Planetlab VM Data

**Received:** 12014, June 2014, Revised 19 July 2014, Accepted 27 July 2014



## 1. Introduction

An ideal way of addressing the energy inefficiency problem is implementing an energy proportional computing system, where energy consumption of the computing resources is proportional to the application workload [21]. This approach is partially implemented through the widely adopted Dynamic Voltage and Frequency Scaling (DVFS) technique. DVFS allows the dynamic adjustment of the voltage and frequency of the CPU based on the current resource demand. As a result, current desktop and server CPUs can consume less than 30% of their peak power in low-activity modes, leading to dynamic power ranges of more than 70% [14]. The reason is that only the CPU supports active low-power modes, whereas other components can only be completely or partially switched off. However, the performance overhead of a transition between the active and inactive modes is substantial. For example, a disk drive in the deep-sleep mode consumes negligible power, but a transition to the active mode

incurs a latency 1000 times higher than the regular access latency [4]. Power inefficiency of the server components in the idle state leads to a narrow overall dynamic power range of a server on the order of 30%. This means that even if a server is completely idle, it still consumes more than 70% of its peak power[4]. Another method is Virtual Machine Consolidation in virtualized environment. Virtualization allows Cloud providers to create multiple VM instances on a single physical server, thus improving the utilization of resources and increasing the Return On Investment (ROI)[4]. Providers and their customers are essential for Cloud computing environments. Therefore, Cloud providers have to deal with the energy-performance trade-off – minimizing energy consumption, while meeting QoS requirements. The scope of this paper is energy-efficient dynamic VM consolidation in Infrastructure as a Service (IaaS) Cloud data centers under QoS constraints.

## 2. Literature Review

### 2.1 VM Consolidation Problem

In recent times, Virtualization has become a key asset in several areas of Computer Science. Advantage of virtualization is that it offers the opportunity to assign multiple Virtual Machine in the same datacenter or in physical computing entity. This process, known as server consolidation, is used by data centers to increase resource utilization and reduce electric power consumption costs [3]. Server consolidation is particularly important when user workloads are unpredictable and need to be revisited periodically. Whenever a user demand changes, VMs can be resized and migrated to other physical servers if necessary. The consolidation process can be performed in a single step using the peak load demands of each workload to configure virtual machine capacities, or reevaluating periodically the workload demand in each virtual machine and performing the required configuration changes. The first approach is known as static consolidation since the virtual machines stay in the same physical servers during their whole lifetime. The utilization of the peak load demand ensures that the virtual machine is not overloaded, however, it can also lead to idleness since the workloads can present variable demand patterns. The second approach is known as dynamic consolidation and usually results in better consolidation, since it dynamically changes virtual machine capacities according to the current workload demands. Here in our work we will use dynamic VM consolidation technique.

### 2.2 VM Consolidation Technique

A Beloglazov et al. [1] [2] [4] considered VM consolidation technique has to deal with four scenarios:

1. Deciding if a host is considered to be under-loaded, so that all VMs should be migrated from it, and the host should be switched to a low-power mode.

2. Deciding if a host is considered to be overloaded, so that some VMs should be migrated from it to other active or reactivated hosts to avoid violating the QoS requirements.

3. Selecting VMs to migrate from an overloaded host.

4. Placing VMs selected for migration on other active or reactivated hosts.

Two major advantages of this approach over other traditional methods are : (1) splitting the problem simplifies the analytic treatment of the sub-problems; and (2) the approach can be implemented in a distributed manner by executing the underload / overload detection and VM selection algorithms on compute hosts, and the VM placement algorithm on replicated controller hosts. Distributed VM consolidation algorithms enable the natural scaling of the system when new compute hosts are added, which is essential for large-scale Cloud providers. So VM consolidation can easily divided into 4 sub problems.

1. Host underload detection.

2. Host overload detection.

3. VM selection.

4. VM placement.

Algorithm needs to be designed separately so that an close to optimal migration can take place to ensure minimum number active physical server and minimum violation of SLA.

### 2.3 Heuristic Methods for Overload Detection and VM Selection

Algorithm needs to be designed in such a way where there will minimum energy consumed and minimum violation of SLA and

efficient VM migration and minimum number active host in a given time. There are considerable number of researches has been conducted for VM consolidation using different types of Heuristics. In [1] [3] Beloglazov et al. proposed different types of threshold that have been calculated through different statistical measures, like: Median Absolute Deviation, The Random Selection Policy, The Maximum Correlation Policy, Local Regression. Here all the sub problems are addressed. Most of the heuristics are designed in such a way that they are adaptive to changes and keep their threshold changing based on different scenario in different time so that they can still provide the functionality and consolidation decision in changed environment. This adaption process allows the system to be dynamic. In the survey papers by Y Li et al., [9] R k Raj et al [10], and J. Sekhar et al [11], different live migration techniques have been discussed which are also based on heuristic approaches, where sub problem 3&4 VM selection and placement are addressed with different threshold which are identified from knowledge or experience. In [4] VM consolidation with migration control is introduced. Here VM with steady usage are not migrated and not steady VMs are migrated to ensure better performance, the migrations are triggered and done by Heuristic approaches. Main advantage of heuristics is that a static and acceptable performance is always can be get with some errors.

### 2.4 Cloud Sim

CloudSim is developed by Clouds lab in the University of Melbourne. Reference [6] [7] [9], describes CloudSim, which provides various functionality of a cloud environment and facilitates cloud simulation. Reference [1] [2] [4] have also used CoudSim for simulation. The main advantage of using cloud SIM is Some Heuristics approaches are already built in the simulation as those heuristics have been in used in the referred work. The main components of CloudSim are datacenter, Virtual Machine (VM) and cloudlet. Cloudlet can be real data from real cloud. The simulator creates Datacenter, virtual Machine and cloudlet on the run based on the defined parameters. When the simulation starts, virtual machines are placed in the datacenter for processing. Sub problems(1-4) are already developed in CloudSim. To develop any further one need to create new class to develop new methods and test it. It has various functionalities. The functionality of interest of this work is energy calculation which is can be derived after simulation and various metrics are generated. Based on those performance of an algorithm or methods can be tested.

## 3. Proposed Model

In this work we have designed three algorithms, by doing amalgamation of heuristic method and migration control. From the above mentioned sub problems, this work is focused on subproblem-3 which is VM selection. VM selection is the immediate step after the host overload is detected. Now decision needs to be made to migrate the VM. Selecting the VM efficiently is an important task and the energy consumption depends on the selection. In [1][2][4] three VM selection techniques have been proposed which are based on various states of VMs. On the other hand Migration control will influence the VM selection mechanism for steady and resource consuming VMs not to be selected. Reason behind is that, a steady VM which is continuously consuming recourses has high probability of consuming resource in the same manner in future. So if we migrate such VM it will also overload the next datacenter in which we are migrating it to. In this proposed model, CPU usage has been considered to identify a VM to be steady or not. There are basically three VM selection mechanism which were discussed in [1][2][4], they are 1) Minimum Migration time, 2) Maximum correlation and 3) Random selection. These mechanisms are modified and equations are also modified to adopt migration control in the policy and algorithms are developed. Proposed VM selection methods are discussed below along with the modified equation and algorithms:

### 3.1 Minimum Migration Time with Migration Control

Minimum Migration time policy migrates the VM which can be migrated within minimum time limit. So this migration time is limited by the memory the VM is using and the bandwidth and migration control is applied.

Let, there are two VMs $x$ and $y$ in host $h$ and $V_h$ be the set of VMs. $CPU_u(x_t)$ is the CPU utilization of time $t$, which is current time. And $CPU_u(x_{t-1})$, $CPU_u(x_{t-2})$ ….. $CPU_u(x_{t-n})$ are the $CPU$ utilization of up to previous n number time frames where Overload detection algorithm was activated and $CPU_{Threshold}$ is the threshold for the algorithm to determine whether the VM is steady or not. t is the time and n is the window and NET is the bandwidth.

At any moment $t$, The MMT with Migration Control policy finds VM $x$ will be selected for migration for the below formulas:

$$x \in V_h \mid \forall y \in V_h, \frac{RAM(x)}{NET_h} \leq \frac{RAM(y)}{NET_h}$$

Only if;

$$\frac{[CPU_u(x_t) + CPU_u(x_{t-1}) + CPU_u(x_{t-2}) + \ldots.. CPU_u(x_{t-n})]}{n+1} < CPU_{thresold} \quad (1)$$

If for every VM *vm*,

$$\forall\, vm \in V_h\,,$$
$$\frac{[CPU_u(vm_t) + CPU_u(vmx_{t-1}) + \ldots.. + CPU_u(vm_{t-n})]}{n+1} \geq CPU_{thresold}$$

Then,

$$x \in V_h \mid \forall\, y \in V_h, \frac{RAM(x)}{NET_h} \leq \frac{RAM(y)}{NET_h} \quad (2)$$

Equation 1 ensures if the VM with minimum RAM will be selected for migration only if the VM satisfies the migration controlled threshold parameter. If the lowest *VM* according to *RAM* does not satisfy it will check the second lowest and in this way all *VM* will be checked to match the criteria.

If no such *VM* is found to have satisfied the migration control threshold parameter from the whole set of VM then according to equation (2) the lowest RAM consuming *VM* is selected for the migration.

**Algorithm 1.1 MMT with Migration Control(mmtmc)**

Pre-conditions: overloaded host *h*, Migration control threshold $CPU_{threshold}$, window size *n*

Post-condition: Virtual Machine to be migrated $VM_m$

1. Input Overloaded *host h* ;
2. $VM_h$ = *getmigratablevm* (*h*);
3. $VM_{hex}$ = *Excludevminmigration* ($VM_h$);
4. *for each VM* $V_i$ *of* $VM_{hex}$
5. $CPU_{mc}$ = *getmcparamfromcpuhistory* ($V_i$);
6. *RAM* ($V_i$) = *getram* ($V_i$);
7. *If RAM* ($V_i$) / $NET_h$ is lowest till now
8. $VM_{lowest} = V_i$ ;
9. i*f* $CPU_{mc} < CPU_{threshold}$ then $VM_m = V_i$ ;
10. End;
11. End;
12. If $VM_m$ is *null*: $VM_m = VM_{lowest}$ ;
13. Return $VM_m$ ;

The Algorithm 1.1 describes how Minimum migration time with migration control (mmtmc) works. The input of the algorithm is the overloaded host which is detected by previous phase: Overload detection. After having the host h at step-1, at step-2, *getmigratablevm* (*h*) function pulls all the *VM* which are currently placed on that host *h*. At step-3, *Excludevminmigration* function excludes all the *VM* which are already in migration for that hot and assigned to $VM_{hex}$ . At step-5, for each *VM* $V_i$ , migration control parameter is generated from the function *getmcparamfromcpuhistory* ($V_i$). This function follows equation 1.

The parameter n is determined through trail and error basis to find out which window works best. At Step-6, current usage of RAM will be fetched for $V_i$ and will check for if the one is the lowest upto now. If this is one is the lowest till now, at step-8, $VM_{lowest}$ will be updated. At step-9, migratable *VM* $VM_m$ will be updated if $CPU_{mc}$ smaller than $CPU_{threshold}$. If all *VM* is greater than the threshold and the lowest mmt *VM* is selected for migration at step-12 and step-13 return the *VM* to be migrated.

**3.2 Maximum Correlation with Migration Control**

Maximum correlation policy [2] [4] works based on the idea that the higher the correlation between the resource usage by applications running on an oversubscribed server, the higher the probability of server overloading. Basically this instructs that higher correlation of CPU usage of one *VM* with other *VM* should be migrated. Migration control is applied with maximum correlation method to identify the migratable *VM*.

Let there are *n* numbers of *VM* and $X_1, X_2, ....X_n$ is the CPU usage of *n VM* which are considered migration. Let *Y* be the *VM* where we want to determine the maximum correlation with $i^{th}$ *VM*. The Augmented to matrix for the rest is denoted by *X* and the $(n-1)$ *x*1 vectored of *Y* is expressed by *y*.

$$X = \begin{bmatrix} 1 & x_{1,1} & \cdots & \cdots & x_{1,n-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & x_{n-1,1} & \cdots & \cdots & x_{n-1,n-1} \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ . \\ . \\ . \\ . \\ y_n \end{bmatrix} \tag{3}$$

A vector of predicted value is denoted by $\hat{y}$.

$$\hat{y} = Xb \quad b = (X^T X)^{-1} X^T y \tag{4}$$

Having found the predicted value the correlation coefficient is:

$$R^2_{Y, X_1, \ldots X_{n-1}} = \frac{\sum_{i=1}^{n} (y_i - m_y)^2 \quad (\hat{y}_i - m_{\hat{y}})^2}{\sum_{i=1}^{n} (y_i - m_y)^2 \sum_{i=1}^{n} (\hat{y}_i - m_{\hat{y}})^2} \tag{5}$$

Here $m_y$ and $m_{\hat{y}}$ are the sample mean of *Y* and $\hat{Y}$. Now the multiple correlation coefficient is found by

$$R^2_{X_i, X_1, \ldots X_{n-i}, X_{n+i} \ldots X_n}.$$

To select a *VM u* by Maximum Correlation with migration Control will be:

$$u \in V_h \mid \forall a \in V_h, R^2_{X_i, X_1, \ldots X_{n-i}, X_{n+i} \ldots X_n} \geq R^2_{X_i, X_1, \ldots X_{n-i}, X_{n+i} \ldots X_n}$$

Only if;

$$\frac{[CPU_u(x_t) + CPU_u(x_{t-1}) + CPU_u(x_{t-2}) + \ldots CPU_u(x_{t-n})]}{(n+1)} < CPU_{thresold} \tag{6}$$

If for every *VM vm*,

$$\forall vm \in V_h,$$
$$\frac{[CPU_u(vm_t) + CPU_u(vmx_{t-1}) + \ldots + CPU_u(vm_{t-n})]}{(n+1)} \geq CPU_{thresold}$$

Then,

$$u \in V_h \mid \forall a \in V_h, R^2_{X_i, X_1, \ldots X_{n-i}, X_{n+i} \ldots X_n} \geq R^2_{X_i, X_1, \ldots X_{n-i}, X_{n+i} \ldots X_n} \tag{7}$$

Equation 6 and 7 express the criteria of MC with migration control.

Equation 6 ensures if the *VM* with minimum correlation will be selected for migration only if the *VM* satisfies the migration controlled threshold parameter. If the lowest *VM* according to correlation does not satisfy it will check the second lowest and in this way all *VM* will be checked to match the criteria.

If no such *VM* is found to have satisfied the migration control threshold parameter from the whole set of *VM* then according to equation (6) the lowest correlation coefficient for a *VM* is selected for the migration.

**Algorithm 1.2 MC with Migration Control(mcmc)**

*Pre-conditions:* overloaded *host h*, Migration control threshold $CPU_{threshold}$,window size *n*

*Post-condition:* Virtual Machine to be migrated $VM_m$

1. Input Overloaded *host h* ;
2. $VM_h$ = *getmigratablevm* (*h*);
3. $VM_{hex}$ = Excludevminmigration ($VM_h$);
4. *Utilm* ($VM_{hex}$)=utilizationmatrix ($VM_{hex}$);
5. *Metric* (*n*) =correlationcoefficient (*Utilm* ($VM_{hex}$));
6. for each metric *i* of Metric (*n*)
7. $CPU_{mc}$ = getmcparamfromcpuhistory ($V_i$);
8. If *i* is lowest till now
9. $VM_{lowest}$ = *getvm* (*i*);
10. if $CPU_{mc} < CPU_{threshold}$ then $VM_m$ = *getvm* (*i*);
11. End;
12. End;
13. If $VM_m$ is null: $VM_m = VM_{lowest}$ ;
14. Return $VM_m$;

**The Algorithm 1.2** Describes how Maximum Correlation with migration control (mcmc) works. The input of the algorithm is the overloaded host which is detected by previous phase: Overload detection. After having the host *h* at step-1, at step-2, *getmigratablevm* (*h*) function pulls all the *VM* which are currently placed on that host *h*. At step-3, *Excludevminmigration* function excludes all the *VM* which are already in migration for that hot and assigned to $VM_{hex}$. at step-4, using utilizationmatrix function utilization matrics is calculated. At step-5, Correclation coefficient matrics is calculated using equation 5. for each I of *Metric* (*n*), migration control parameter is generated from the function *getmcparamfromcpuhistory* ($V_i$). This function follows equation 6. The parameter *n* is determined through trail and error basis to find out which window works best. At Step-8, correlation coefficient is checked whether it is the lowest upto now or not. If this is one is the lowest till now, at step-8, $VM_{lowest}$ will be updated. At step-10, migratable *VM* $VM_m$ will be updated if $CPU_{mc}$ smaller than $CPU_{threshold}$. If all *VM* is greater than the threshold and the lowest *mmt VM* is selected for migration at step-13 and step-14 return the *VM* to be migrated.

### 3.3 Random Selection with Migration Control

Random selection policy selects a random *VM* to be migrated for the overloaded host. In the *RS* with migration(rsmc) control, migration control criteria is included to impose control for steady *VM* not to be migrated.

The rsmc selects a *VM* to be migrated from the overloaded host according to a uniformly distributed discrete random variable whose value index is a set of *VM*s $V_h$ allocated to host *h*. So the equations are:

$$u \overset{d}{\Leftarrow} R(0, |V_h|)$$

Only if;

$$\frac{[CPU_u(x_t) + CPU_u(x_{t-1}) + CPU_u(x_{t-2}) + \ldots.. CPU_u(x_{t-n})]}{(n+1)} < CPU_{thresold:} \quad (8)$$

If for every VM *vm*,

$$\forall\, vm \in V_h\,,$$

$$\frac{[CPU_u(vm_t) + CPU_u(vmx_{t-1}) + \ldots.. + CPU_u(vm_{t\text{-}n})]}{(n+1)} \geq CPU_{thresold}$$

Then,

$$u \overset{d}{\Leftarrow} R\,(0, |V_h|) \quad (9)$$

Equation 8 ensures the randomly selected *VM* will be selected for migration only if that *VM* satisfies the migration controlled threshold parameter. Once not found the that *VM* will be deleted from the *VM* list and will again search for the randomly selected *VM* and will check for the migration control parameter..

If no such *VM* is found to have satisfied the migration control threshold parameter from the whole set of *VM* then according to equation (9) One Random *VM* will be selected for migration.

**Algorithm 1.3 RS with Migration Control(rsmc)**

**Pre-conditions:** overloaded *host h*, Migration control threshold $CPU_{threshold}$, window size *n*

**Post-condition:** Virtual Machine to be migrated $VM_m$

1. Input Overloaded *host h ;*
2. $VM_h$ = *getmigratablevm* (*h*);
3. $VM_{hex}$ = *Excludevminmigration* ($VM_h$);
4. $VM_{Random} = VM_{hex\,;}$
5. While $VM_{Random}$ is not *NULL*
6. $V_R$= *Random* ($VM_{Random}$);
7. $CPU_{mc}$= *getmcparamfromcpuhistory* ($V_R$)*;*
8. *if* $CPU_{mc} < CPU_{threshold}$ *then* $VM_m = V_R$;
9. *Else* $VM_{Random}$= *deletevm* ($V_R$);
10. End;
11. *If* $VM_m$ *is null*: $VM_m$= *Random* ($VM_{hex}$);
12. *Return* $VM_m$;

**The Algorithm 1.3** Describes how Random selection with migration control (rsmc) works. The input of the algorithm is the overloaded host which is detected by previous phase: Overload detection. After having the host *h* at step-1, at step-2, *getmigratablevm* (*h*) function pulls all the VM which are currently placed on that host *h*. At step-3, *Excludevminmigration* function excludes all the VM which are already in migration for that hot and assigned to $VM_{hex}$. At step-4, $VM_{hex}$ is copied to $VM_{Random}$ for further use. At step-5, loop started for $VM_{Random}$ until it is NULL. One Random *VM* $V_R$ is selected using Random function. At step-7,control parameter is generated from the function *getmcparamfromcpuhistory* ($V_R$). This function follows equation 8. The parameter *n* is determined through trail and error basis to find out which window works best. At Step-8,

migratable *VM* $VM_m$ will be updated if $CPU_{mc}$ smaller than $CPU_{threshold}$. At step-9, $V_R$ deleted from the list if it does not satisfy the condition and the loop continues and try to find another random *VM* until the list is empty. *VM* is selected selected randomly for migration at step-12 if it was not selected earlier. In step-13 return the *VM* to be migrated.

**4. Experimental Results**

For simulation we have used Cloudsim 3.0.3 to evaluate our proposed algorithms. Cloudsim offers wide range of functionality and is better than other simulation program like SimGrid or Gangsim. Cloudsim not only provide the built in simulation classes, there are also some heuristic energy aware *VM* consolidation algorithm designed from reference [1]. Performance metrics are also formulized and can be customized. Energy consumption is calculated based on the HP ProLiant G4 and HP ProLiant G5 from CPU usage and power consumption.

| **Server** | 0% | 20% | 40% | 60% | 80% | 100% |
|---|---|---|---|---|---|---|
| HP ProLiant G4 | 86 | 92.6 | 99 | 106 | 112 | 117 |
| HP ProLiant G4 | 93.7 | 101 | 110 | 121 | 129 | 135 |

Table 1. Power consumption based CPU load of server

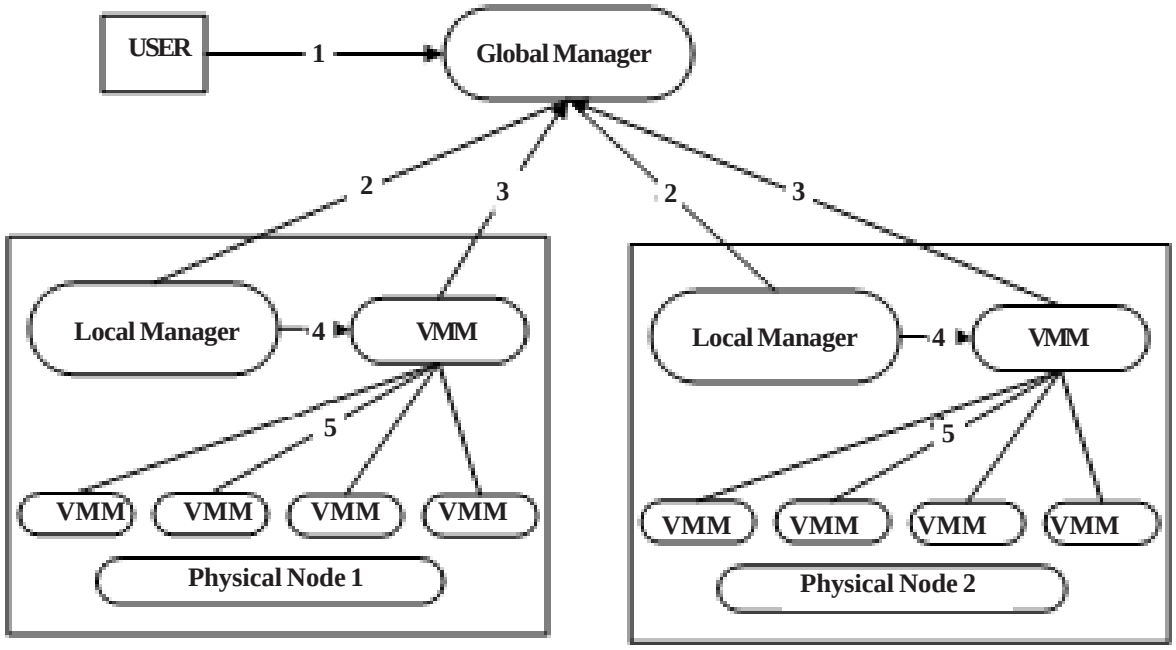


Figure 1. The System Architecture

The system is made with local and global managers [2]. The local managers reside on each node as a module of the *VMM*. Their objective is the continuous monitoring of the node's CPU utilization, resizing the *VMs* according to their resource needs, and deciding when and which *VMs* should to be migrated from the node (4). The global manager resides on the master node and collects information from the local managers to maintain the overall view of the utilization of resources (2). The global manager issues commands for the optimization of the *VM* placement (3). *MMs* perform actual resizing and migration of *VMs* as well as changes in power modes of the nodes (5).

In this simulation we have considered 800 heterogeneous physical node, half of which are HP ProLiant G4 and rest are HP ProLiant G5 servers. These servers are assigned with 1860MIPS and 2660 MIPS for each core of G4 and G5 servers. And network bandwidth is considered as 1GB/s. The *VM* which were created were single core. *VM* were assigned 4 types. High-CPU Medium Instance (2500 MIPS, 0.85 GB); Extra Large Instance (2000 MIPS, 3.75 GB); Small Instance (1000 MIPS, 1.7 GB); and Micro Instance (500 MIPS, 613 MB) [4].

To evaluate the performance of the proposed methods, it is necessary to use workload traces from real system. In this simulation we have used Planetlab workload traces. This trace is taken from 2011 March and April. The data contains CPU utilization data of thousand *VM* of 500 different location. In every *VM* file, there 288 CPU utilization data means, CPU utilization after every 5 min. So, when Simulation started, *VM* is created and assigned in different datacenter, after assigning in random fashion, Overload detection algorithm is run, if found the *VM* selection algorithm is executed and migrated to less utilized server. After handling overloaded server, under loaded servers are detected and migrate the *VM* to other datacenter without making that overloaded. After migrating the datacenter is put to sleep mode. In every iteration in 5 minites, energy consumption and *VM* migration, SLA and host shutdown is counted and provide a final result at the end of the simulation.

There are five built-in overload detection algorithm. 1) A Static CPU Utilization Threshold (THR) 2) Adaptive Median Absolute Deviation(MAD) 3) Adaptive Interquartile Range (IQR) 4) Local Regression(LR) 5) Robust local Regression(LRR) and there are three VM selection algorithm built in, 1) Random Selection (RS) 2) Maximum Correlation (MC) 3) Minimum Migration time (MMT). These policys will be be compared to proposed three policies 1) Random selection with Migration Contorl (RSMC) 2) Maximum Correlation with migration Control (MCMC) and 3) Minimum Migration time with migration control (MMTMC). Experimental result will be shown by making pair between Overload detection and *VM* selection policies. In the simulation for proposed policies, window size is used as 5, means migration control will work for last five iteration and $CPU_{threshold}$ is used for *mmtmc* = 40, *mcmc* = 70 and *rsmc* = 85.

Comparison with previous heuristics and with new proposed method is given below. For each policy 5 metrics are compared, 1)Energy Consumpion(kWh) 2) Number of *VM* migration 3) *ESLAV* = Energy consumption * SLA 4) Average SLA violation 5) Number of host shutdown. For example, *iqr _ mmt* is compared with *iqr_mmtmc* Here iqr is the overload detection algorithm and *mmt/mmtmc* means minimum migration time and minimum migration time with migration control. 15 combinations with new method will be compared to 15 combinations of old methods.

**4.1 Maximum Correlation with Migration Control**

Energy consumption is compared for mc and mcmc Figure 2. From graph we can find that iqr_mcmc, lr_mcmc, lrr_mcmc and thr _ mcmc consume less energy than the previous heuristics.

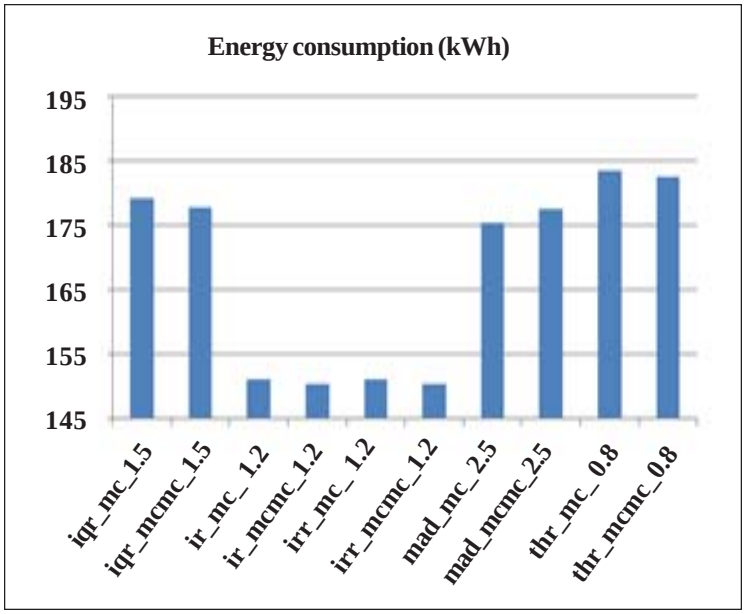


Figure 2. Engergy Consuption comparison between mc and *mcmc*

Number of *VM* migration is compared for mc and mcmc Figure 3. From graph we can find that *iqr_mcmc, lr_mcmc, lrr_mcmc* policy incurs less number of *VM* migrations than the previous heuristics.

ESLAV is compared for mc and mcmc Figure 4. From graph we can find that *iqr_mcmc, lr_mcmc, lrr_mcmc* and *mad_mcmc* policy produce less *ESLAV* value than the previous heuristics.

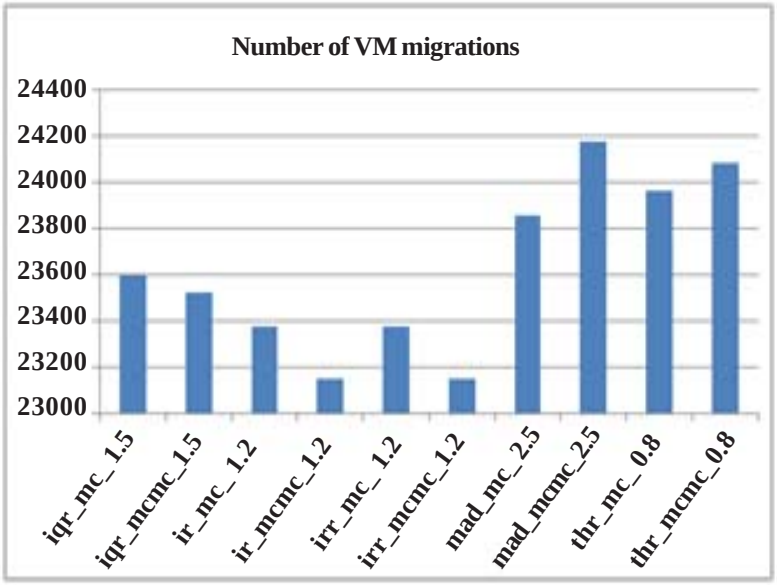


Figure 3. Number of VM migration comparison between *mc* and *mcmc*

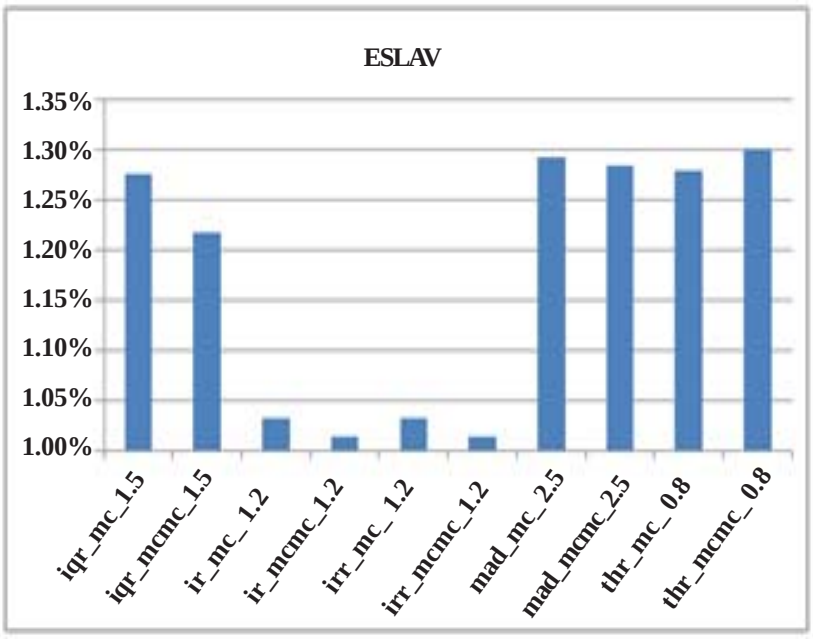


Figure 4. ESLAV comparison between *mc* and *mcmc*

Average SLA violation is compared for mc and mcmc Figure 5. From graph we can find that that iqr_mcmc, lr_mcmc, lrr_mcmc and thr_mcmc policy incurs has less Average SLA violation than the previous heuristics.

Number of host shutdown is compared for mc and mcmc Figure 6. From graph we can find that iqr_mcmc, lr_mcmc, lrr_mcmc and thr_mcmc policy incurs less number of host shutdown than the previous heuristics.

**4.2 Randowm Selection with Migration Control**

Energy consumption is compared for mc and mcmc Figure 7. From graph we can find that iqr_rsmc, lrr_rsmc and mad_rsmc consume less engery than the previous heuristics.

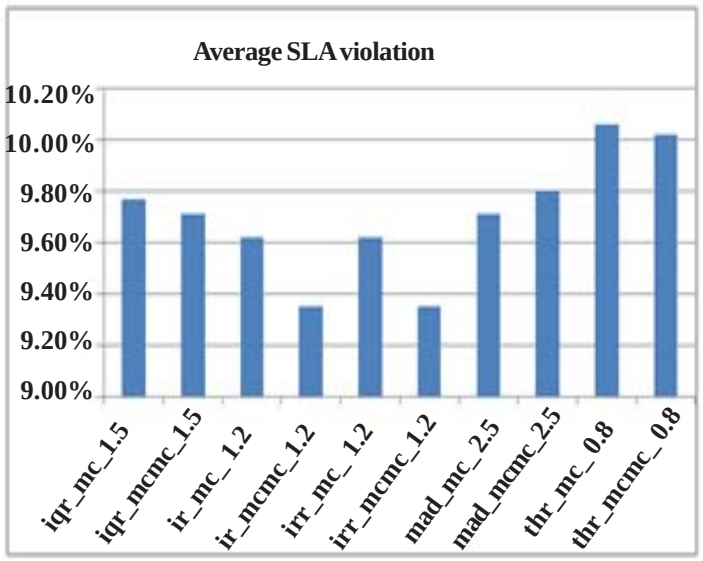


Figure 5. Average SLA violation comparison between *mc* and *mcmc*

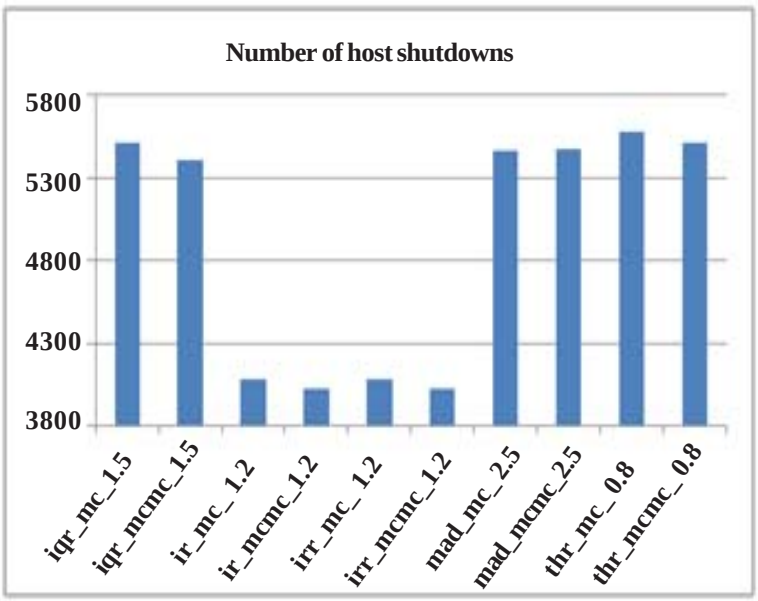


Figure 6. Number of host shutdown between *mc* and *mcmc*

Number of *VM* migration is compared for *rs* and *rsmc* Figure 8. From graph we can find that *iqr_rsmc*, *lrr_rsmc*, *mad_rsmc* policy incurs less number of *VM* migration than the previous heuristics.

ESLAV is compared for rs and rsmc Figure 9. From graph we can find that *iqr_rsmc*, *lrr_rsmc* and *mad_rsmc* policy produce less ESLAV value than the previous heuristics.

Average SLA violation is compared for rs and rsmc Figure 10. From graph we can find that that *iqr_rsmc* has similar like previous heuristics and *lr_rsmc*, *lrr_rsmc* and *thr_rsmc* policy incurs has less Average SLA violation than the previous heuristics.

Number of host shutdown is compared for rs and rsmc Figure 11. From graph we can find that i*qr_rsmc* and *lrr_rsmc* policy incurs less number of host shutdown than the previous heuristics.

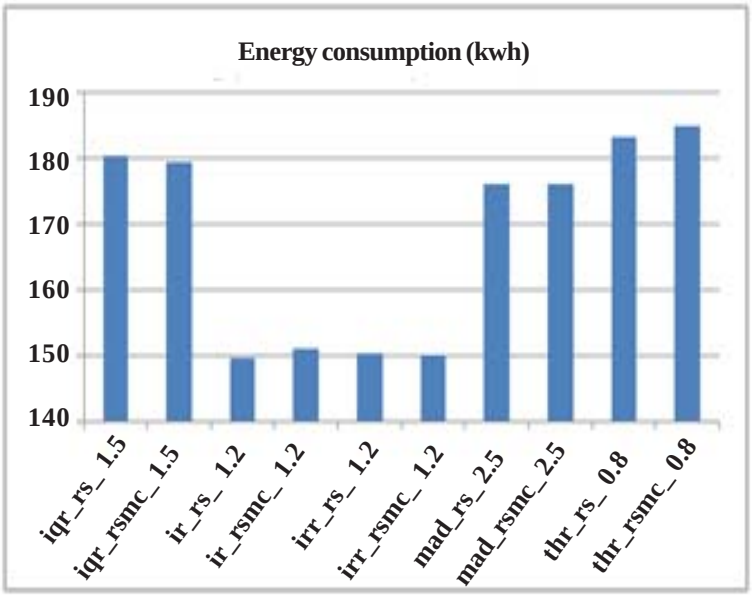


Figure 7. Engergy Consuption comparison between *rs* and *rsmc*

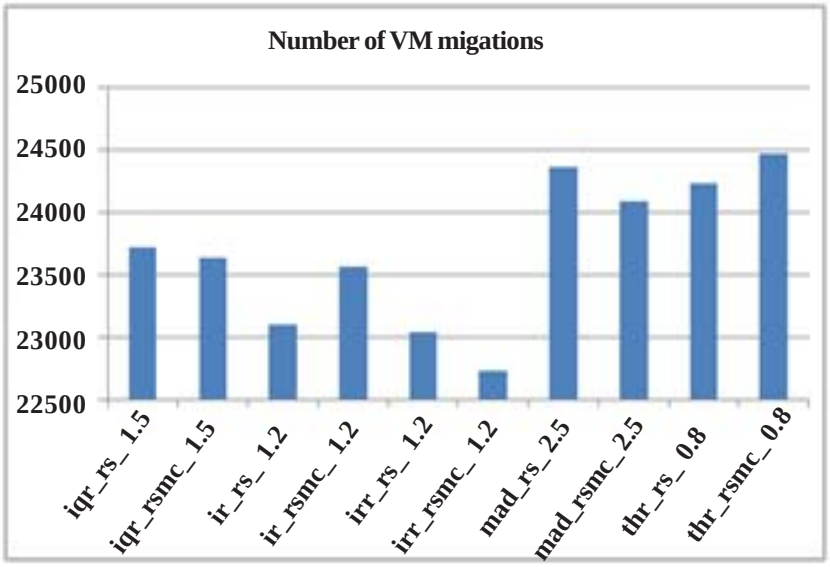


Figure 8. Number of VM migration comparison between *rs* and *rsmc*

**4.3 Minimum Migration Time with Migration Control**

Energy consumption is compared for mc and mcmc Figure 12. From graph we can find that iqr_rsmc, lrr_mcmc and mad_rsmc consume less engery than the previous heuristics.

Number of VM migration is compared for mmt and mmtmc Figure 13. From graph we can find that all new policy incurs less number of VM migration than the previous heuristics.

ESLAV is compared for mmt and mmtmc Figure 14. From graph we can find that iqr_mmtmc, lr_mmtmc policy produce almost same ESLAV value with the previous heuristics.

Average SLA violation is compared for mmt and mmtmc Figure 15. From graph we can find that all policies have less Average SLA violation than the previous heuristics.

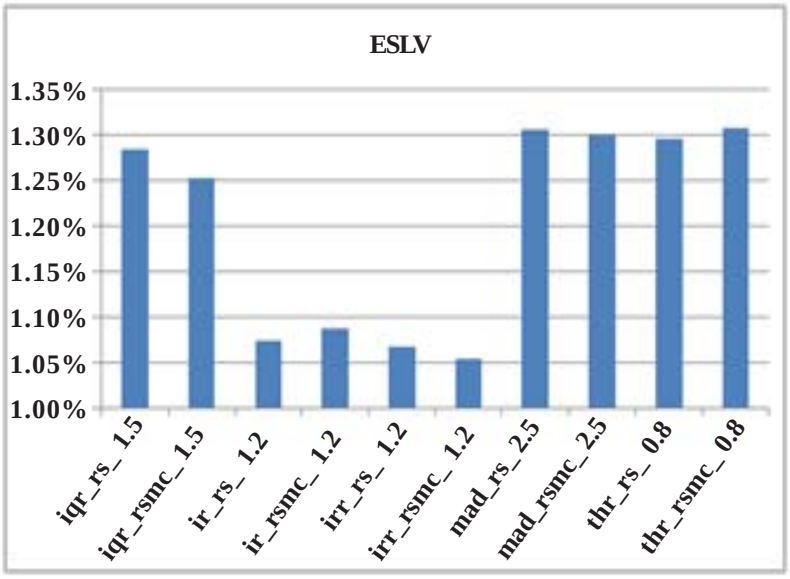


Figure 9. ESLAV comparison between *rs* and *rsmc*

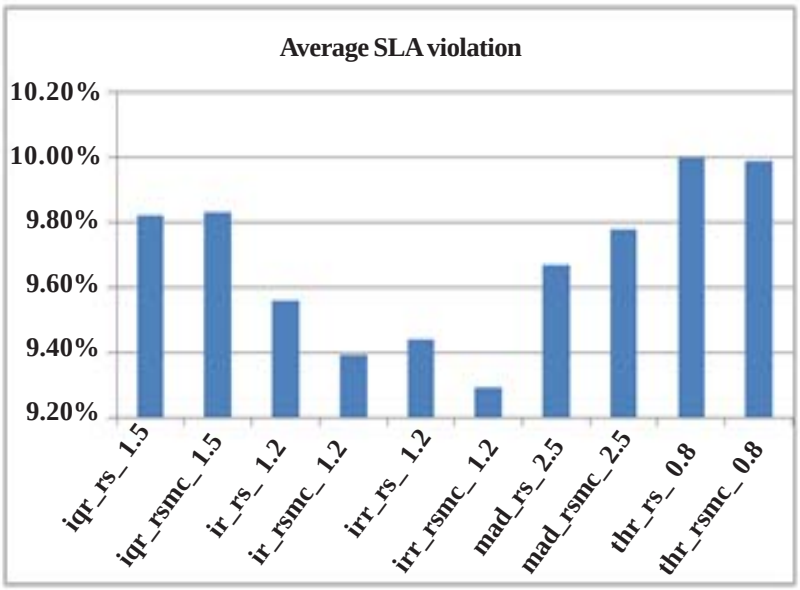


Figure 10. Average SLA violation between *rs* and *rsmc*

Number of host shutdown is compared for mmt and mmtmc Figure 16. From graph we can find that all mmtmc policy incurs less number of host shutdown than the previous heuristics.

**4.4 Result Summary**

The result can be summarized with the following observation:

• For metric Energy consumption, it is found that, out of 15 combination 12 combination outperform the previous methods resulting energy saving.

• Number of *VM* migration also had improvement on 11 combinations than non migration control policy which will definitely reduce traffic of Cloud network.

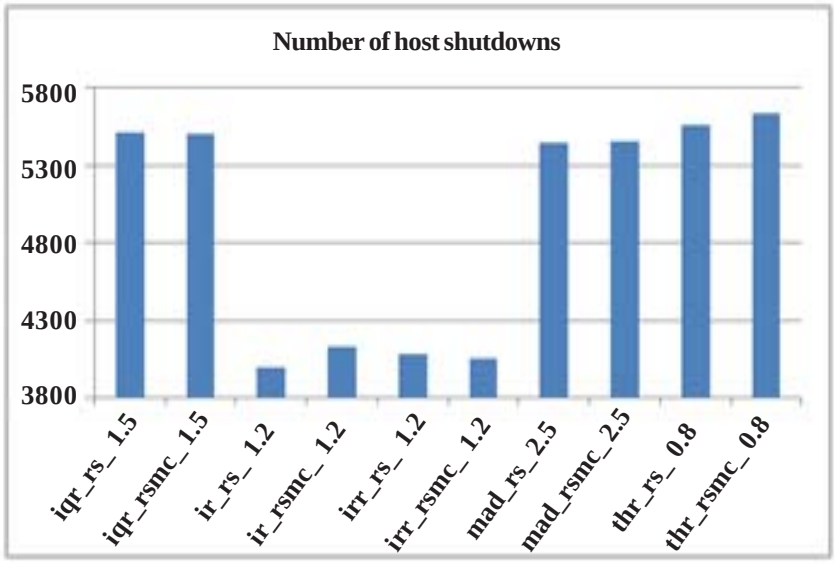


Figure 11. Number of host shutdowns comparison between *rs* and *rsmc*

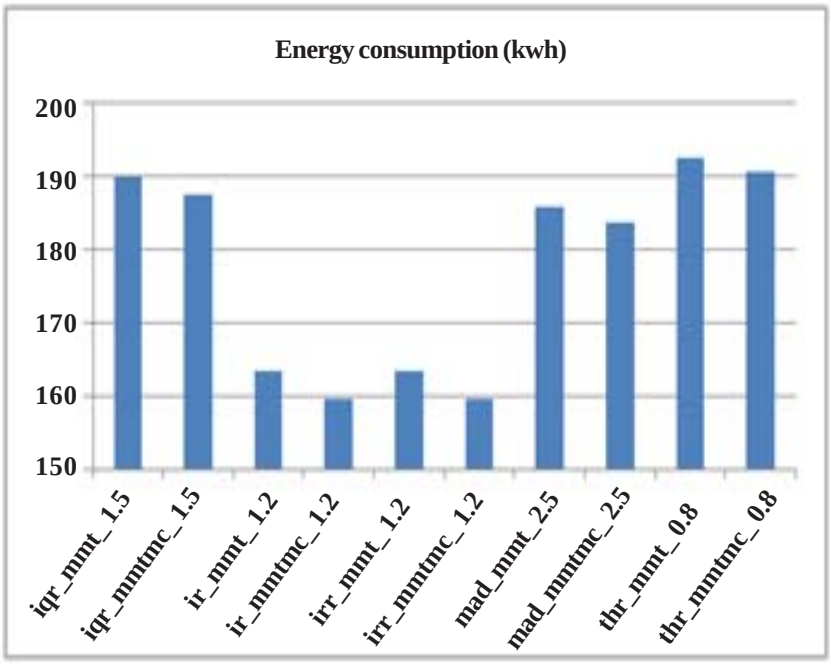


Figure 12. Energy Consuption comparison between *mmt* and *mmtmc*

• ESLAV metric is the product of energy consumption and sla violation. In 7 combinations ESLAV found improved compared to previous method. And 2 have similar result.

• Average SLA violation is also improved. Out of 15, 12 methods incur less Average SLA violation than previous method.

• If number of host shutdown decreases along with power consumption, it means VM migration was efficient. Out of 15, 12 methods showed improvement compared to old method.

• The method MMTMC showed improvement in every metric except ESLAV where two methods were similar to old one.

• Improvement found for 10 combinations in 4 metric. MCMC(3), RSMC(2) and MMTMC(5)

• IQR_MCMC, LR_MCMC, LRR_MCMC and LRR_RSMC showed improvement in 5 metric. IQR_MMTMC and LR_MMTMC had significant improvement in 4 metric and almost similar values in ESLAV.

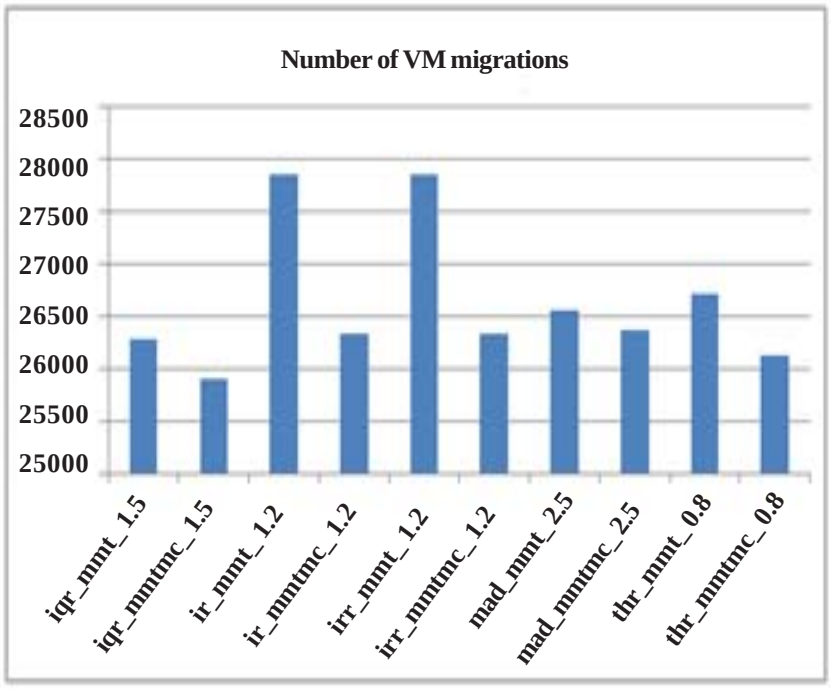


Figure 13. Number of VM migration comparison between *mmt* and *mmtmc*

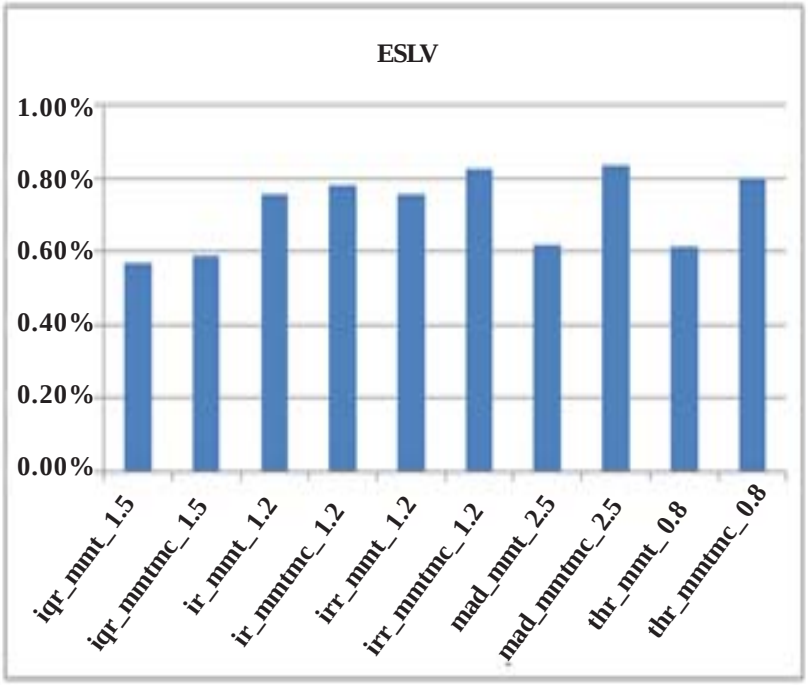


Figure 14. ESLAV comparison between *mmt* and *mmtmc*

**5. Conclusion and Future Work**

Migration control in Energy aware VM consolidation not only saves energy but it shows improvement in VM migration number which will reduce network traffic and also host shutdown is decreased in most of the combination which is also a positive side. Frequent host shutdown can be problematic in long run. Moreover, the simulation was done for one day workload trace of planetlab from 2011, month wise or year wise impact will be lot higher.

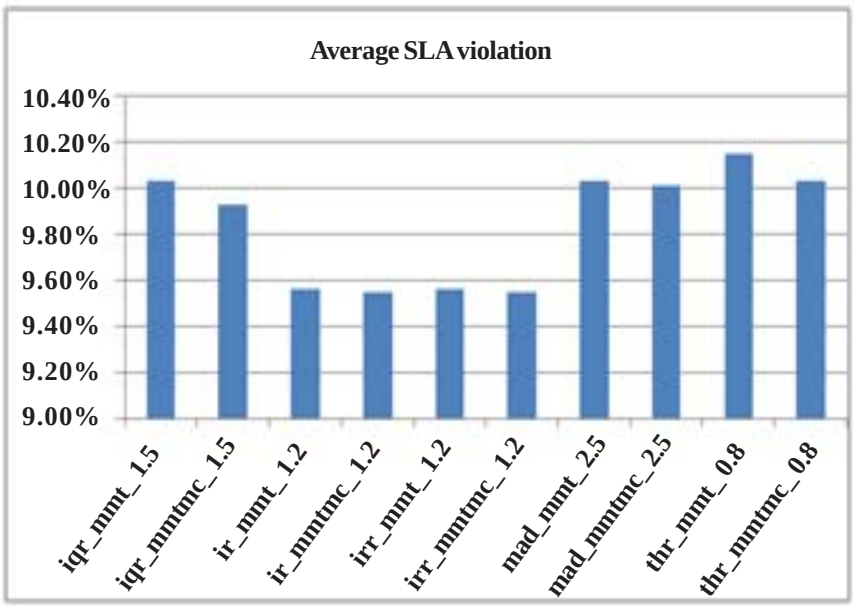


Figure 15. Average SLA violation comparison between mmt and mmtmc

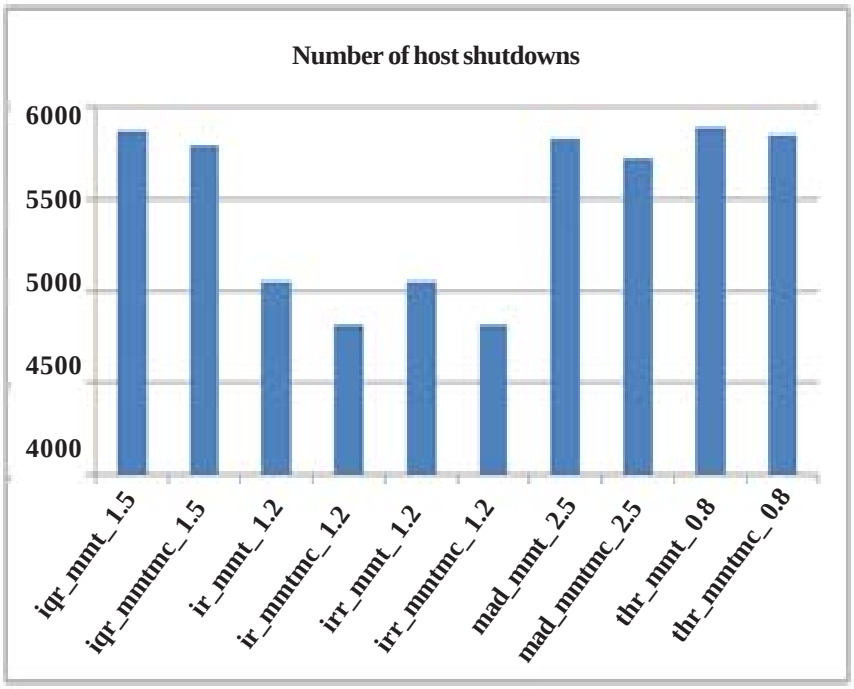


Figure 16. Number of host shutdowns comparison between mmt and mmtmc

For the future work we can work for a better overload detection and overload prediction based on real workload traces and see the result. VM placement and underload handling can also be interesting topic of work.

**References**

[1] Beloglazov, Anton., Abawajy, Jemal., Buyya, Rajkumar. (2012). Energy-Aware Resource Allocation Heuristics or Efficient Management of Data Centers for Cloud Computing, *Future Generation Computer Systems* (FGCS), 28 (5) 755-768, Elsevier Science, The Netherlands.

[2] Beloglazov, Anton., Buyya, Rajkumar. (2012). Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers, Concurrency and Computation: Practice and Experience (CCPE), 24 (13) 397-1420, John Wiley & Sons, Ltd, USA.

[3] Ferreto, T., Netto, M., Calheiros, R., Rose, C. D. (2011). Server consolidation with migration control for virtualized data centers, *Future Generation Computer Systems*, 27 (8) 1027- 1034, October.

[4] Beloglazov, Anton. (2013). Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing, University of Melbourne PhD Thesis.

[5] James Broberg, Andrzej Goscinski, Rajkumar Buyya, M. Cloud Computing: Principles and Paradigms, Chapter 5, 6 and 16. Link: http://as.wiley.com/WileyCDA/WileyTitle/productCd-0470887990.html

[6] Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms, *Software: Practice and Experience*, 41 (1) 23–50.

[7] Calheiros Rodrigo, N., Rajiv Ranjan., Anton Beloglazov., César De Rose, A. F., Buyya, Rajkumar. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, *Software—Practice & Experience*, 41 (1) 23-50, January.

[8] Cloud SIM http://code.google.com/p/cloudsim/, retrived on 20$^{th}$. August, 2014.

[9] Li, Yunfa., Li, Wanqing., Jiang, Congfeng. (2010). A Survey of Virtual Machine System: Current Technology and Future Trends, Third International Symposium on Electronic Commerce and Security, p.332-336, Guangzhou, China.

[10] Raj, Rakhi K., Leelipushpam. P., Getzi Jeba. (2012). Live Virtual Machine Migration Techniques-A Survery, I*nternational Journal of Engineering Research & Technology* (*IJERT*) 1 (7) September.

[11] Sekhar, Jyothi., Jeba, Getzi., Durga. S. (2012). A survery on energy efficient server consolidation through VM live migration, *International Journal of Advances in Engineering & Technology*, November.

[12] Prevost, Nagothu., Kelley, Jamshidi. (2011). Prediction of Cloud Data Center Networks Loads Using Stochastic and Neural Models,6$^{th}$ International Conference on System of Systems Engineering (SoSE), p. 276-281.

[13] Di, Sheng., Kondo, Derrick., Cirne,Walfredo. (2012). Host load prediction in a Google compute cloud with a Bayesian model, *In*: Proceedings of the International Conference on High Performance Computing, Networking, *Storage and Analysis* (SC 2012), Article (21) 1-11.

[14] Barroso, L. A., Holzle, U. (2007). The casse for energy-proportional computing, *Computer,* 40 (12) 33–37.