Annales Mathematicae et Informaticae
***46** (2016) pp. 217–241*
http://ami.ektf.hu

Evolutionary computing based QoS oriented energy efficient VM consolidation scheme for large scale cloud data centers using random work load bench

Perla Ravi Theja*, SK. Khadar Babu

[a]VIT University, Vellore Campus
Vellore - 632 014, Tamilnadu, India
ravithejaperla9048@gmail.com
khadar.babu36@gmail.com

Submitted December 22, 2015 — Accepted June 7, 2016

Abstract

In order to assess the performance of an approach, it is unavoidable to inspect the performance with distinct datasets with diverse characteristics. In this paper we had assessed the system performance with random workbench datasets. A-GA (Adaptive Genetic Algorithm) based consolidation technique has been compared with other consolidation techniques including dynamic CPU utilization techniques, VM (Virtual Machine) selection and placement policies. The proposed consolidation system had exhibited better results in terms of energy conservation, minimal Service Level Agreement (SLA) violation and Quality of Service (QoS) assurance.

MSC: 68U20; 68U99; 65K10; 65K05

1. Introduction

As computing requirements is changing in a huge rate in all-most all the bossiness's, there requires optimal utilization of resources which yields in saving costs, which motivated everyone to make use of Cloud computing. The key issues with

*Corresponding author

Cloud computing are energy consumption, SLA violation, Cloud Network Security and Privacy. As per report from Natural Resources Defence Council (NRDC), in 2013 data centers in USA used around 91 billion kilowatt-hours of electricity that is matching to the annual output of 34 large (500-megawatt) coal-fired power plants. By 2020 this demand may increase to roughly 140 billion kilowatt-hours that is matching to annual output of 50 power plants, costing American businesses 13 billion USD annually in electricity bills and emitting nearly 100 million metric tons of carbon pollution per year [21]. In this paper we are concentrating on energy conservation, reducing SLA violation and providing QoS assurance by implementing dynamic thresholding based resource utilization detection approach for real time cloud systems and by developing an evolutionary computing paradigm based A-GA for VM Placement Policy in VM Consolidation application. In this paper we exhibited performance analysis of the proposed system with other approaches having diverse CPU utilization approaches i.e. (Static Threshold (THR), Local Regression (LR), Robust Local Regression (LRR), Median Absolute Deviation (MAD), and Inter Quartile Range (IQR) etc.), VM selection policies i.e. (Minimum Migration Time (MMT) Maximum Correlation (MC) and Random Selection (RS)) and VM placement policies i.e. (Power Aware Best Fit Decreasing (PABFD) and Ant Colony Optimization (ACO)).

2. Related work

Lot of researches have made research on VM consolidation approaches [11, 16, 17, 23] for QoS and energy optimization. Coming to overload detection in data centers, researchers primarily made use of the static threshold approach [3], where they used overall CPU utilization and defined static thresholds. As dynamic prediction is better than Static prediction researches directed towards dynamic cloud pattern and workloads [4]. On the basis of the historical pattern of CPU utilization, Buyya et al [4] came up with a dynamic threshold scheme for upper and lower bound estimation. Regression based CPU utilization was suggested in [9, 10], where the CPU utilization was estimated at host nodes. Various researches used linear regression and the K-nearest neighbour regression schemes for approximating the data retrieved throughout the VMs lifetime. They gave importance for SLA optimization. Later, bin packing problem was considered in [1, 24, 25] for efficient VM consolidation. All most all existing approaches have used PABFD based consolidation [23]. Lot of algorithms tried to maximize number of host's shutdown by providing server consolidation. Multiple researches tried to provide dynamic schemes for optimal consolidation [6, 7, 13].

Various researches tends towards green cloud computing using bio-inspired technique [2]. Resource provision through ant colony model was inspected in [14]. For the purpose of VM placement, ACO and GA were suggested in [5] and [8] respectively. Disintegrated ACO placement scheme was put forwarded in [12]. Resource placement module in virtualization using GA was inspected in [15, 22, 26]. Great number of current researchers had concentrated either on VM selection or VM

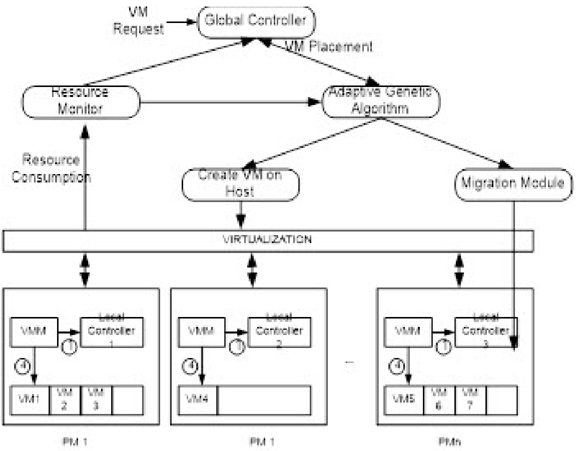

Figure 1: Proposed system architecture

placement. Real work load based A-GA VM placement schemes are used in various researches [18, 19, 20].

3. Our contribution

In our earlier work we validated the optimal efficiency of our proposed A-GA based consolidation with MMT selection policy and combined IQR and LRR CPU prediction schemes by simulation using real world work load from more than a thousand planet-lab virtual machines [18, 19, 20]. In this paper, we performed comparative analysis between different combinations of host overload detection algorithms, VM selection policies and VM placement policies for random workload traces.

3.1. Cloud model

As a part of large scale cloud infrastructure we considered varied physical machines (PMs) or hosts, which are attributed in terms of its CPU consumption and Millions Instructions per Second (MIPS). Storage will be allotted using Storage area network which supports VM migration. Here VMs seek MIPS which will be allotted to carry work.

The VM consolidation minimizing number of active hosts by allotting VM's to run on each single PM so as to conserve energy and to facilitate optimal resource consumption. This process involves SLA Violation and downtime which need to be minimized by applying optimal algorithms. Proposed VM Consolidation scheme as shown in Figure 1 involves two controllers i.e. Global and Local Controllers. Local Controller (LC) which periodically check the status of PM whether it is Overloaded/Underloaded. If it is Overloaded it will report to Virtual Machine

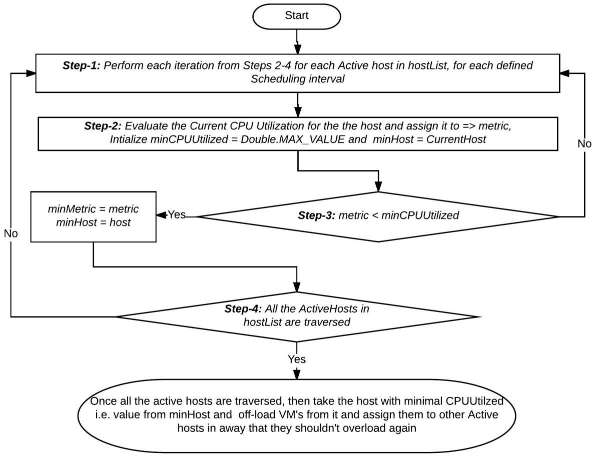
Start
Step-1: Perform each iteration from Steps 2-4 for each Active host in hostList, for each defined Scheduling interval
Step-2: Evaluate the Current CPU Utilization for the the host and assign it to => metric, Intialize minCPUUtilized = Double.MAX_VALUE and minHost = CurrentHost
No
minMetric = metric
minHost = host
Yes
Step-3: metric < minCPUUtilized
No
Step-4: All the ActiveHosts in hostList are traversed
Yes
Once all the active hosts are traversed, then take the host with minimal CPUUtilzed i.e. value from minHost and off-load VM's from it and assign them to other Active hosts in away that they shouldn't overload again

Figure 2: Data Flow diagram for Host under load detection algorithm

Manager (VMM), which in turn triggers VM selection policy and decides which VMs that need to be offloaded to avoid SLA performance degradation and reports to Global Controller (GC) which places these VMs onto other active hosts in a way that they shouldn't overload again using VM placement polices. If it is underloaded then in same way VMM triggers VM selection policy and GC uses VM placement policy to place these VM's.

3.1.1. Host under load and overload detection

The following data flow diagram i.e. Figure 2, illustrates, how the Host under detection algorithm is carried-out. The outcome of this algorithm is minimum CPU utilized host, which will be slept (turned-off) resulting in energy conservation. To investigate whether the host is overloaded or not, we make use of host overload detection algorithms. Here we considered various algorithms i.e. (Median Absolute Deviation (MAD), Static Threshold (THR), Inter Quartile Range (IQR), Local Regression (LR) and Local Robust Regression (LRR)) for this purpose. The process-flow of each algorithm is illustrated completely in terms of data flow diagrams in Figures 3 to 7 respectively as mentioned below:

To detect the overloaded host node, each host node initializes an overload detection scheme intermittently to perform VM de-consolidation, thus enabling SLA violation avoidance. CPU utilization of the host node has been used for identifying overloaded hosts. Unlike conventional schemes based on static threshold, in this paper we have developed a dynamic threshold based adaptive CPU utilization and overload detection scheme. It enables the proposed system to behave in real time scenario where there is highly fluctuating resource utilization. It adjusts resource

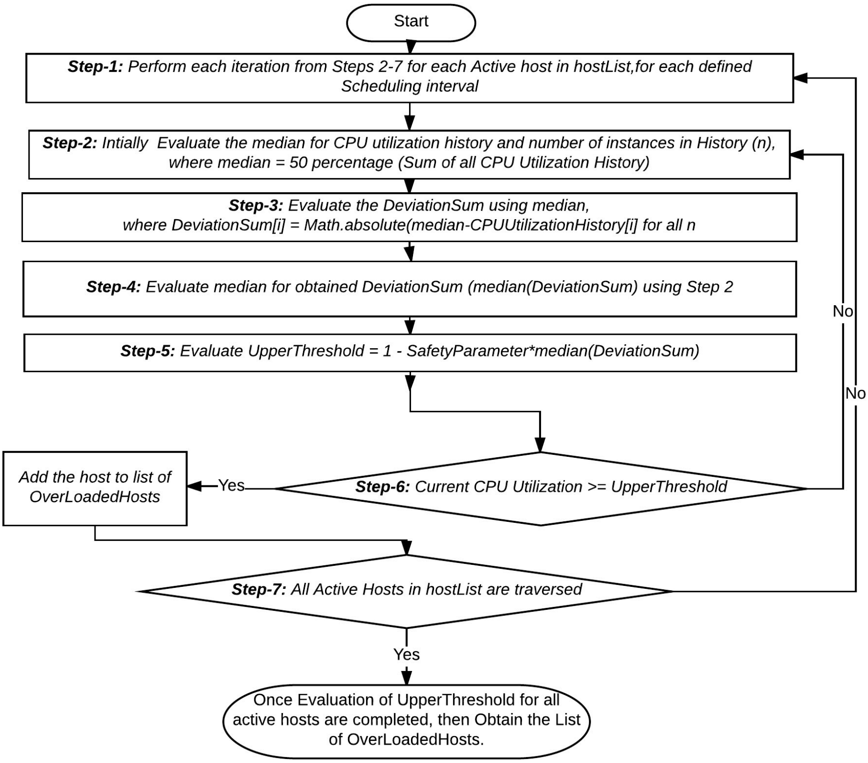
Start
Step-1: *Perform each iteration from Steps 2-7 for each Active host in hostList,for each defined Scheduling interval*
Step-2: *Intially Evaluate the median for CPU utilization history and number of instances in History (n), where median = 50 percentage (Sum of all CPU Utilization History)*
Step-3: *Evaluate the DeviationSum using median, where DeviationSum[i] = Math.absolute(median-CPUUtilizationHistory[i] for all n*
Step-4: *Evaluate median for obtained DeviationSum (median(DeviationSum) using Step 2*
Step-5: *Evaluate UpperThreshold = 1 - SafetyParameter*median(DeviationSum)*
No
No
Add the host to list of OverLoadedHosts
Yes
Step-6: *Current CPU Utilization >= UpperThreshold*
Step-7: *All Active Hosts in hostList are traversed*
Yes
Once Evaluation of UpperThreshold for all active hosts are completed, then Obtain the List of OverLoadedHosts.

Figure 3: Data Flow diagram for Median Absolute Deviation (MAD) algorithm

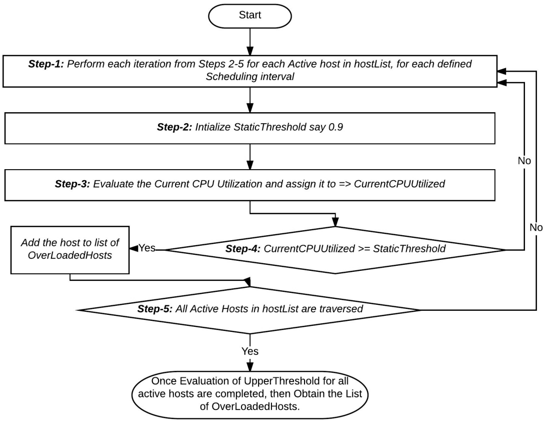
Start
Step-1: *Perform each iteration from Steps 2-5 for each Active host in hostList, for each defined Scheduling interval*
Step-2: *Intialize StaticThreshold say 0.9*
Step-3: *Evaluate the Current CPU Utilization and assign it to => CurrentCPUUtilized*
No
No
Add the host to list of OverLoadedHosts
Yes
Step-4: *CurrentCPUUtilized >= StaticThreshold*
Step-5: *All Active Hosts in hostList are traversed*
Yes
Once Evaluation of UpperThreshold for all active hosts are completed, then Obtain the List of OverLoadedHosts.

Figure 4: Data Flow diagram for Static Threshold (THR) algorithm

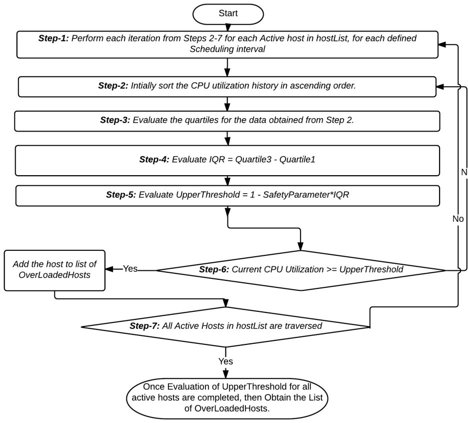
Start
Step-1: Perform each iteration from Steps 2-7 for each Active host in hostList, for each defined Scheduling interval
Step-2: Intially sort the CPU utilization history in ascending order.
Step-3: Evaluate the quartiles for the data obtained from Step 2.
Step-4: Evaluate IQR = Quartile3 - Quartile1
Step-5: Evaluate UpperThreshold = 1 - SafetyParameter*IQR
No
No
Add the host to list of OverLoadedHosts
Yes
Step-6: Current CPU Utilization >= UpperThreshold
Step-7: All Active Hosts in hostList are traversed
Yes
Once Evaluation of UpperThreshold for all active hosts are completed, then Obtain the List of OverLoadedHosts.

Figure 5: Data Flow diagram for Inter Quartile Range (IQR) algorithm

utilization threshold based on the variation in CPU utilization and utility map. It can be observed that higher deviation might even result into 100% CPU utilization that signifies higher overloading probability. To enable dynamic threshold detection scheme, we have used Interquartile Range (IQR) and robust local regression (LRR) algorithm. In our proposed model, the resource utilization has been examined at the interval of 5 minute and for each odd iteration, IQR algorithm has been used, while LRR has been scheduled for even iterations. Such novelty intends to employ major advantages of both approaches. A brief of IQR is given as follows:
IQR is a statistical dispersion technique which is equivalent to the differences between the third and first quartile. To estimate dynamic CPU utilization threshold, the following equation has been used.

$$IQR = Q_3 - Q_1, T_m = 1 - s.IQR \tag{3.1}$$

Where s represents the safety parameter that states the maximum extent of the tolerability of a host node in Cloud environment and its lower value signifies the higher tolerance to the fluctuation in the CPU utilization. Here we have used $s = 1.2$, and it can be changed to examine the optimal performance. In addition to IQR, we have used Loess concept [18] to derive the LRR algorithm, which has been employed for fitting a trend polynomial to the earlier k observations for the CPU utilization called utilization map. For initial recent observations, it is retrieved as

$$\hat{g}(x) = (\hat{a}) + \hat{b}x \tag{3.2}$$

It is further used for calculating the next observation $\hat{g}(x_{k+1})$. To offload some VMs from an overloaded host node, the following conditions have been fulfilled:

$$s.\hat{g}(x_{k+1}) \geq 1, \quad x_{k+1} - x_k \leq t_m \tag{3.3}$$

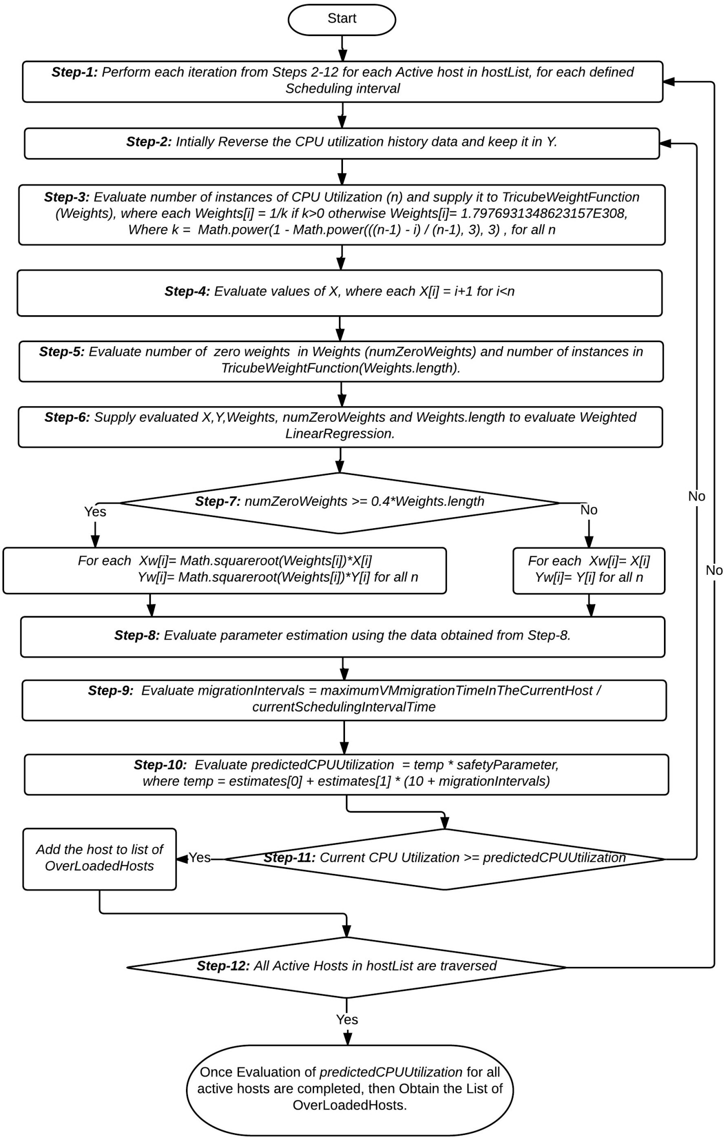
Start
Step-1: Perform each iteration from Steps 2-12 for each Active host in hostList, for each defined Scheduling interval
Step-2: Intially Reverse the CPU utilization history data and keep it in Y.
Step-3: Evaluate number of instances of CPU Utilization (n) and supply it to TricubeWeightFunction (Weights), where each Weights[i] = 1/k if k>0 otherwise Weights[i]= 1.7976931348623157E308, Where k = Math.power(1 - Math.power(((n-1) - i) / (n-1), 3), 3) , for all n
Step-4: Evaluate values of X, where each X[i] = i+1 for i<n
Step-5: Evaluate number of zero weights in Weights (numZeroWeights) and number of instances in TricubeWeightFunction(Weights.length).
Step-6: Supply evaluated X,Y,Weights, numZeroWeights and Weights.length to evaluate Weighted LinearRegression.
Step-7: numZeroWeights >= 0.4*Weights.length
Yes
No
For each Xw[i]= Math.squareroot(Weights[i])*X[i] Yw[i]= Math.squareroot(Weights[i])*Y[i] for all n
For each Xw[i]= X[i] Yw[i]= Y[i] for all n
No
No
Step-8: Evaluate parameter estimation using the data obtained from Step-8.
Step-9: Evaluate migrationIntervals = maximumVMmigrationTimeInTheCurrentHost / currentSchedulingIntervalTime
Step-10: Evaluate predictedCPUUtilization = temp * safetyParameter, where temp = estimates[0] + estimates[1] * (10 + migrationIntervals)
Add the host to list of OverLoadedHosts
Yes
Step-11: Current CPU Utilization >= predictedCPUUtilization
Step-12: All Active Hosts in hostList are traversed
Yes
Once Evaluation of predictedCPUUtilization for all active hosts are completed, then Obtain the List of OverLoadedHosts.

Figure 6: Data Flow diagram for Local Regression (LR) algorithm

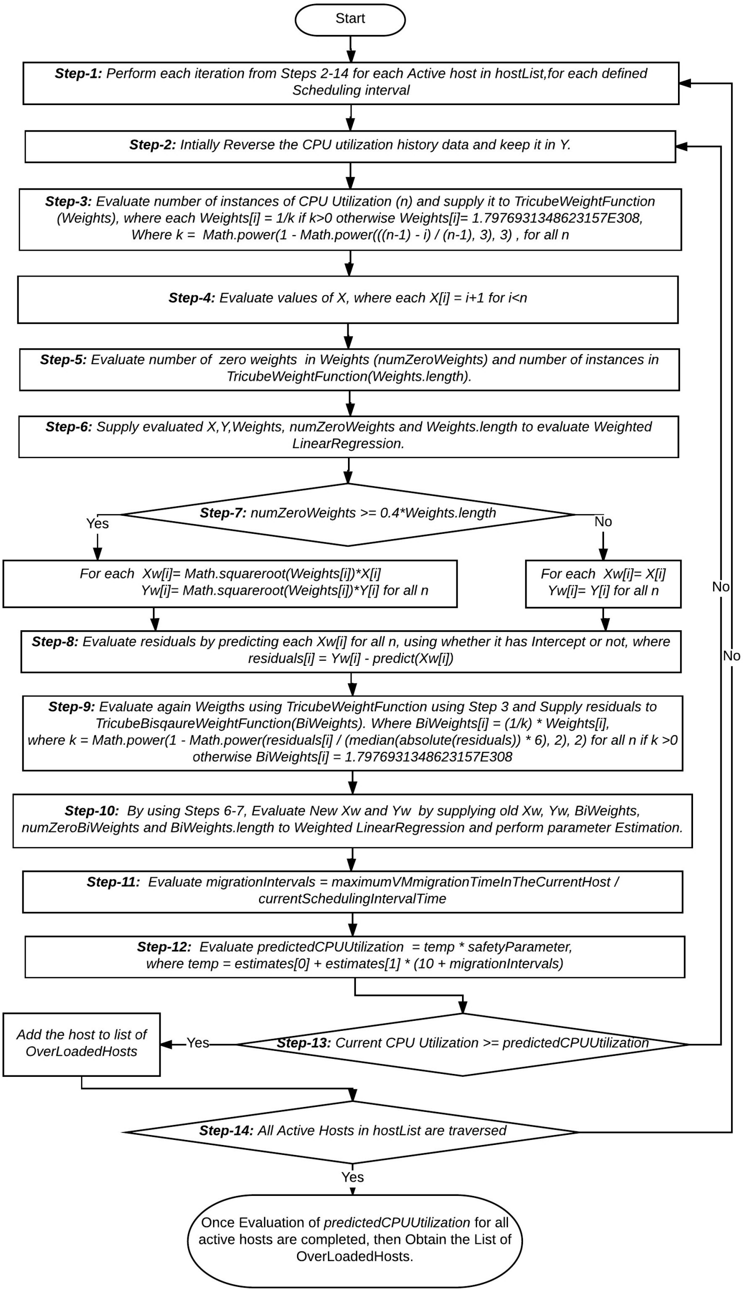
Start
Step-1: Perform each iteration from Steps 2-14 for each Active host in hostList,for each defined Scheduling interval
Step-2: Intially Reverse the CPU utilization history data and keep it in Y.
Step-3: Evaluate number of instances of CPU Utilization (n) and supply it to TricubeWeightFunction (Weights), where each Weights[i] = 1/k if k>0 otherwise Weights[i]= 1.7976931348623157E308, Where k = Math.power(1 - Math.power(((n-1) - i) / (n-1), 3), 3) , for all n
Step-4: Evaluate values of X, where each X[i] = i+1 for i<n
Step-5: Evaluate number of zero weights in Weights (numZeroWeights) and number of instances in TricubeWeightFunction(Weights.length).
Step-6: Supply evaluated X,Y,Weights, numZeroWeights and Weights.length to evaluate Weighted LinearRegression.
Step-7: numZeroWeights >= 0.4*Weights.length
Yes
No
For each Xw[i]= Math.squareroot(Weights[i])*X[i] Yw[i]= Math.squareroot(Weights[i])*Y[i] for all n
For each Xw[i]= X[i] Yw[i]= Y[i] for all n
No
No
Step-8: Evaluate residuals by predicting each Xw[i] for all n, using whether it has Intercept or not, where residuals[i] = Yw[i] - predict(Xw[i])
Step-9: Evaluate again Weigths using TricubeWeightFunction using Step 3 and Supply residuals to TricubeBisqaureWeightFunction(BiWeights). Where BiWeights[i] = (1/k) * Weights[i], where k = Math.power(1 - Math.power(residuals[i] / (median(absolute(residuals)) * 6), 2), 2) for all n if k >0 otherwise BiWeights[i] = 1.7976931348623157E308
Step-10: By using Steps 6-7, Evaluate New Xw and Yw by supplying old Xw, Yw, BiWeights, numZeroBiWeights and BiWeights.length to Weighted LinearRegression and perform parameter Estimation.
Step-11: Evaluate migrationIntervals = maximumVMmigrationTimeInTheCurrentHost / currentSchedulingIntervalTime
Step-12: Evaluate predictedCPUUtilization = temp * safetyParameter, where temp = estimates[0] + estimates[1] * (10 + migrationIntervals)
Add the host to list of OverLoadedHosts
Yes
Step-13: Current CPU Utilization >= predictedCPUUtilization
Step-14: All Active Hosts in hostList are traversed
Yes
Once Evaluation of predictedCPUUtilization for all active hosts are completed, then Obtain the List of OverLoadedHosts.

Figure 7: Data Flow diagram for Local Robust Regression (LRR) algorithm

Where $s \in \mathbb{R}^+$ is the safety parameter that signifies the maximum capability or tolerability of a host node and t_m represents the maximum time required for migrating a VM from the overloaded host. The traditional Loess concept [18] is found vulnerable to the outliers caused due to leptokurtic or heavy-tailed distributions. To alleviate this problem, we have modified the Loess concept [18] to bisquare from the conventional least-squares (LR) approach. Thus we have LRR approach iteratively for estimating the initial fitting and tricube weight function has been used for dynamic weight estimation. The fitting parameter has been retrieved at x_i to get optimal value by means of $\hat{y_i}$. Thus, the final residual value is $\epsilon_i = y_i - \hat{y_i}$. Furthermore, the final retrieved value (x_i, y_i) has been assigned a robustness factor $\mathbb{R}_i$ that primarily depends on the magnitude of ϵ_i. Mathematically, R_i is obtained as

$$\mathbb{R}_i = \mathbb{B}\left(\frac{\hat{\epsilon}_i}{6L_{MAD}}\right) \tag{3.4}$$

Where $\mathbb{B}(.)$ gives the bisquare weight function and L_{MAD} represents the Median Absolute Deviation (MAD) for the least square fit. Mathematically:

$$B(.) = \begin{cases} (1-u^2)^2 & \text{if} |u| < 1, \\ 0 & \text{otherwise.} \end{cases}$$

In this paper, we assigned $\mathbb{R}_i$ for each observation (5 minute), where L_{MAD} has been obtained as:

$$L_{MAD} = median|\hat{\epsilon}_i| \tag{3.5}$$

Using equation (3.3), the next observation has been obtained for the estimated trend line, where observing any inequalities, the host can be identified as overloaded.

3.1.2. VM selection policy

In this phase, the VM selection takes place where it is intended to select the VM which should be migrated to minimize overhead from overloaded host. Estimating the dynamic CPU utilization threshold, VM selection has been performed that offloads host for avoiding SLA violation and unwanted energy consumption caused due to overload. In this paper, we have examined three different selection policies; the Minimum Migration Time (MMT), Maximum Correlation (MC) and Random Selection (RS) policy. A brief discussion of the implemented VM selection policies with respective data flow diagrams are reprsented in Figures 8 to 10 as mentioned below:

1. **Minimum Migration Time(MMT) Policy**
   Once assessing the host's CPU utilization levels and identifying any probable or overloaded host, VMs selection algorithm performs offloading of that host node to avoid any probability of SLA violation. The developed MMT selection policy performs migration of only those VMs (v) which requires minimal migration time than the other. In this paper, the migration time has been

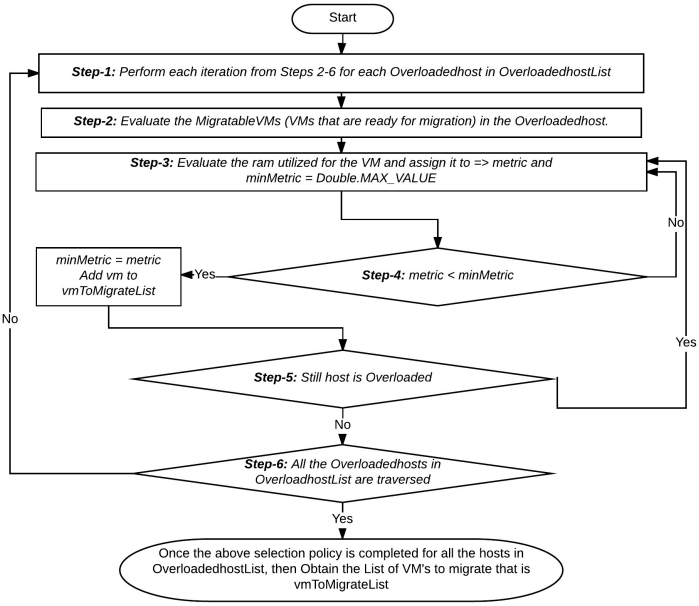
Start
Step-1: Perform each iteration from Steps 2-6 for each Overloadedhost in OverloadedhostList
Step-2: Evaluate the MigratableVMs (VMs that are ready for migration) in the Overloadedhost.
Step-3: Evaluate the ram utilized for the VM and assign it to => metric and minMetric = Double.MAX_VALUE
minMetric = metric Add vm to vmToMigrateList
Yes
Step-4: metric < minMetric
No
No
Yes
Step-5: Still host is Overloaded
No
Step-6: All the Overloadedhosts in OverloadhostList are traversed
Yes
Once the above selection policy is completed for all the hosts in OverloadedhostList, then Obtain the List of VM's to migrate that is vmToMigrateList

Figure 8: Data Flow diagram for Minimum Migration Time (MMT) policy

estimated in terms of the resource, RAM being used by VM divided by the supplementary network bandwidth available for hostj. Let $V_(j)$ be a set of VMs connected with the hostj. Therefore, the VM to be migrated is selected based on the following condition.

$$v \in V_j | \forall_a \in V_j, \frac{RAM_u(v)}{NET_j} \leq \frac{RAM_u(a)}{NET_j} \tag{3.6}$$

Where $RAM_u(a)$ depicts the amount of RAM currently being used by VMa; and NET_j refers the bandwidth available for migration from the host$j$.

2. **Maximum Correlation (MC) Policy**
In case of maximum correlation [19][20] policy, it is assumed that the higher correlation between the CPU utilization by VMs connected to certain host signifies higher overloading probability. In MC policy, VMs having highest correlation for the CPU utilization are needed to be migrated from the current host to assist energy conservation and SLA violation avoidance. We have used the concept of multiple correlation coefficients (MCC) for estimating the intra-VM correlation and respective CPU utilization. The used MCC coefficients are in relation to the squared correlation between the real and the predicted values of the dependent variable. In fact, it can be interpreted as the fraction of variance of the dependent variable elucidated by the associated

independent variables.

Let $X_1, X_2, \cdots, X_n$ be the CPU utilization of n VMs connected to a host and Y be the specific VM to be migrated, where $(n-1)$ are the random independent variables. Here Y being the VM to be migrated, be the dependent variable. In our proposed MC policy, we have calculated the correlation between Y and $(n-1)$. The obtained augmented matrix $(n-1) \times n$ encompasses the observed instances or the values of $(n-1)$, indicated as X. Similarly, the observation vector or mapped vector $(n-1) \times 1$ of the dependent variable Y be y. Thus, the overall observation vector or the mapping vectors for the VM's CPU utilization can be given as follows:

$$x = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1,1} & \cdots & x_{n-1,n-1} \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} \tag{3.7}$$

Observing equation (3.7), it can be found that the first column of X is all 1 (for all instances), therefore it can be considered as an augmented matrix. The predicted values of the VM's CPU utilization or Y can be presented as $\hat{Y}$, which can be obtained by $\hat{Y} = Xb$, where $b = (X^T X)^{-1} X^T Y$. Obtaining the predicted values, MCC coefficients, the final correlation $R^2_{Y,1,\cdots,n-1}$ has been obtained as

$$R^2_{Y,X_1,\cdots,X_{n-1}} = \frac{\sum_{i=1}^{n}(Y_i - m_Y)^2(\hat{Y}_i - m_{\hat{Y}})^2}{\sum_{i=1}^{n}(Y_i - m_Y)^2 \sum_{i=1}^{n}(\hat{Y}_i - m_{\hat{Y}})^{2'}} \tag{3.8}$$

Where m_Y and m_Y give the observation means of Y and $\hat{Y}$ respectively. In MC based VM selection, the MCCs for all mapped instances X_i have been obtained as $R^2_{X_i,X_1,\cdots,X_{i-1},X_{i+1},...,X_n}$. Finally, based on correlation value, Eq. (3.9) has been used to select a specific VM to be migrated:

$$\begin{aligned} v \in VM_j | \forall_a & \\ \in V_j, R^2_{X_{vm},X_1,\cdots,X_{vm-1},X_{v+1}X_n} & \qquad (3.9) \\ \geq R^2_{X_v,X_1,\cdots,X_{a-1},X_{a+1},\cdots,X_n}. & \qquad (3.10) \end{aligned}$$

In addition to the MMT and MC selection policy, we have also examined an algorithm called random selection (RS) policy.

3. **Random Selection Policy**

In Random Selection policy, a VM is randomly selected for migration from the host node as per a uniformly distributed discrete random variable $A = U(0|V_j)$, whose values signify a set of VMs, V_j placed at jth host. Since VM consolidation is a bin packing problem and therefore, an optimal approach for placement is of great significance to ensure minimal downtime, energy consumption and probable SLA violation. The following section discusses the proposed evolutionary computing based VM placement approach.

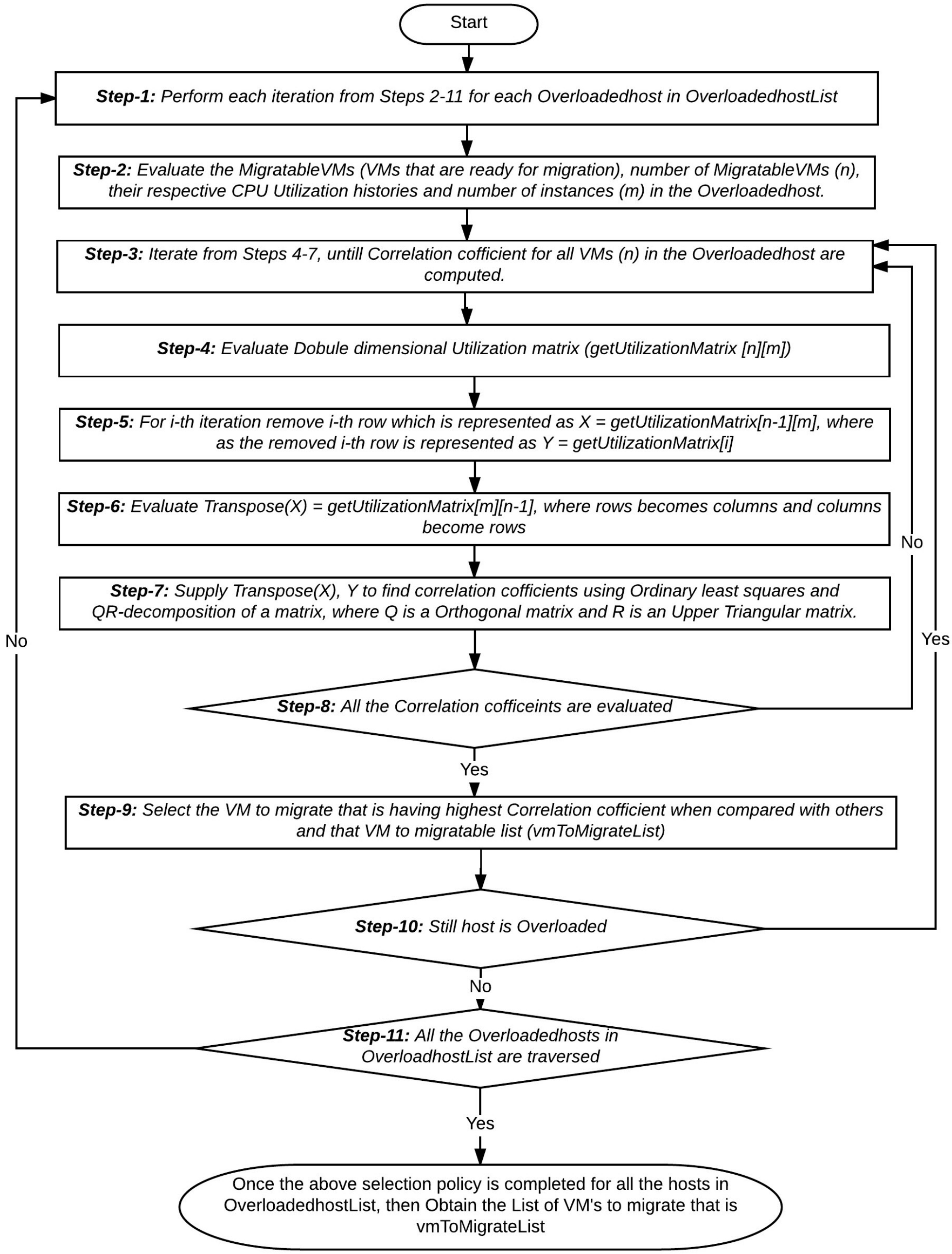
Start
Step-1: Perform each iteration from Steps 2-11 for each Overloadedhost in OverloadedhostList
Step-2: Evaluate the MigratableVMs (VMs that are ready for migration), number of MigratableVMs (n), their respective CPU Utilization histories and number of instances (m) in the Overloadedhost.
Step-3: Iterate from Steps 4-7, untill Correlation cofficient for all VMs (n) in the Overloadedhost are computed.
Step-4: Evaluate Dobule dimensional Utilization matrix (getUtilizationMatrix [n][m])
Step-5: For i-th iteration remove i-th row which is represented as X = getUtilizationMatrix[n-1][m], where as the removed i-th row is represented as Y = getUtilizationMatrix[i]
Step-6: Evaluate Transpose(X) = getUtilizationMatrix[m][n-1], where rows becomes columns and columns become rows
Step-7: Supply Transpose(X), Y to find correlation cofficients using Ordinary least squares and QR-decomposition of a matrix, where Q is a Orthogonal matrix and R is an Upper Triangular matrix.
No
No
Yes
Step-8: All the Correlation cofficeints are evaluated
Yes
Step-9: Select the VM to migrate that is having highest Correlation cofficient when compared with others and that VM to migratable list (vmToMigrateList)
Step-10: Still host is Overloaded
No
Step-11: All the Overloadedhosts in OverloadhostList are traversed
Yes
Once the above selection policy is completed for all the hosts in OverloadedhostList, then Obtain the List of VM's to migrate that is vmToMigrateList

Figure 9: Data Flow diagram for Maximum Correlation (MC) policy

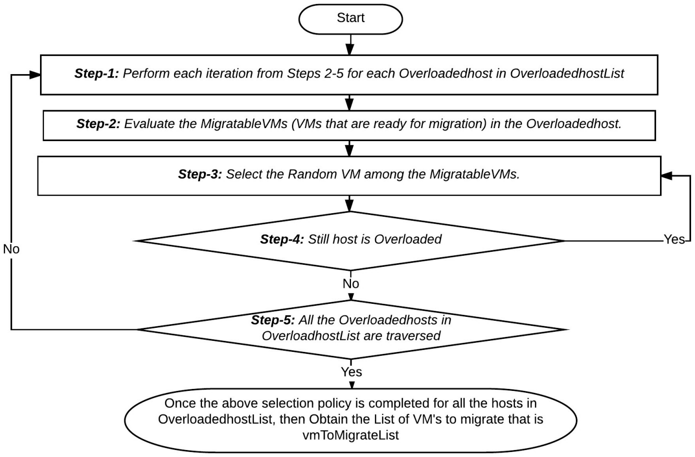
Start
Step-1: Perform each iteration from Steps 2-5 for each Overloadedhost in OverloadedhostList
Step-2: Evaluate the MigratableVMs (VMs that are ready for migration) in the Overloadedhost.
Step-3: Select the Random VM among the MigratableVMs.
Step-4: Still host is Overloaded
Yes
No
No
Step-5: All the Overloadedhosts in OverloadhostList are traversed
Yes
Once the above selection policy is completed for all the hosts in OverloadedhostList, then Obtain the List of VM's to migrate that is vmToMigrateList

Figure 10: Data Flow diagram for Random Selection (RS) policy

3.1.3. VM placement policy

VM placement can be stated to be a problem of bin packing that encompasses bins, items and prices as the three parameters, where bins represent the host nodes, VMs represent the items to be allocated, bin size refers the available resource on the host node; and the resource or CPU power consumed by a host is stated in terms of price. In bin packing, it is intended to accommodate as much as VMs that makes the overall scenario NP-hard. In order to deal with such non-convexity problem, certain heuristic approach or evolutionary computing scheme can be the potential solution. In this paper, we have proposed Adaptive Genetic Algorithm (A-GA) as VM placement policy. We have interfaced A-GA with CloudSim simulator comprising multiple host nodes, and VMs in the data center. In our proposed model, each host is equipped with one or multiple processing elements (PE). The executing VMs on the hosts have one or multiple running Cloudlets. In simulation model, the user requests have been stated in terms of Cloudlets, where the needed processing power for each Cloudlet has been defined in terms of Million Instructions Per Second (MIPS). In the proposed placement policy, the scheduler considers all hosts, VMs and VM maps as input and generates mapping for nodes, where it divides overall MIPS into different components like hosts and VMs running in parallel. The functional discussion of the proposed A-GA scheme is given in (Figure 13).

1. **A-GA Based VM Placement**
   As depicted in (Figure 1), the proposed A-GA processes VM scheduling based on the resource utilization information provided by local controller (LC) and the upper threshold value estimated by dynamic threshold estimation scheme. We have considered upper threshold so as to satisfy transient variations of resource demand by different VMs on a host node. The CPU utilization pattern

or history of VMs has been considered for VMs placement onto destination host node. Considering a large scale cloud infrastructure or data center, the time efficient and effective placement scheduling is of great significance. In this research we have intended to reduce the number of VM migration and migration time, so as to enable energy efficient and QoS oriented consolidation mechanism. In addition, we have scheduled the system to enable maximum host shut down so as to conserve energy. In this paper, placing or allocating VM on certain host, our algorithm estimates the energy of the data center and accordingly performs further scheduling to minimize energy consumption.

At first, the proposed A-GA algorithm initializes a definite set of population where individual host is a tree comprising global controller as its root, hosts are the next level nodes and VMs are the child nodes (Figure 1). It calculates the total energy consumption and CPU utilization for each mapping in the deployed cloud center. Here, the VM mapping history also known as utilization pattern, allocated VMs and their resource utilization mapping, future mapping for VMs based dynamic utilization, hosts and its available resource availability etc. have been used as population. The precise discussion of the proposed system is given in (Figure 13). From these chromosomes, our proposed A-GA algorithm initially selects two VM mappings with minimal energy values on which the initial genetic operators (crossover p_c and mutation probability p_m) are applied. Thus, the mapping obtained for VMs onto the host nodes is added to the overall population based on the fitness values. In our proposed A-GA based VM placement policy, p_c selects the host with the best CPU utilization based on the previous VMs mapping. Here p_c and p_m try to reduce host nodes by means of SWITCHING OFF or turning it into SLEEP MODE. Here, it should be noted that unlike conventional genetic approach (i.e., A-GA), we have applied adaptive genetic parameter selection, where these variables are updated dynamically after every iteration, till stopping criteria (100 generations) is obtained.

Consider, the host nodes in data centers be $PM = pm_1, pm_2, pm_3, \cdots, p_{m_m}$ and pm_i be ith host node, where $(1 \leq i \leq m)$. Similarly, VMs in the network be $VM_i = vm_1, vm_2, vm_3, \cdots, vm_{n,i}$, which are connected to the ith host. Consider $vm_{(j,i)}$ be the ith VM on jth PM. The variable $x_{(i,j)}$ signifies whether ith VM is placed on host j or not. Let $P_{(r,i)}$ be the resource capacity r (CPU utilization) on jth host node. The resource needed by ith VM is $v_(r, j)$. Thus, overall load on j th host node would be the sum of all resource needed by all VMs running over it. Consider, T be the duration of past observations, thus the sub-intervals can be obtained by dividing T into $(q-1)$ sub intervals such that $T = [(t_2 - t_1)(t_3 - t_2)(t_q - t_(q-1))]$. The slot $(t_k - t_(k-1))$ is the time periodk. In such manner, for periodk, we have estimated the CPU utilization at a host $(CPU_(i, Util)(k))$ using following equation:

$$CPU_{(i,Util)}(k) = \sum_{j=1}^{n} \frac{vm_{CPU,j}}{pm_(CPU, j)} \tag{3.11}$$

Where, k represents the duration for which the CPU utilization has to be retrieved. Finally, the average CPU utilization at a host node has been obtained as:

$$pm_(i, AvgUtil) = \sum_{t-t_k}^{t_k-n} \frac{pm_(i, Util)(t)}{(q-1)} \tag{3.12}$$

Where $(q-1)$ represents the total number of sub intervals in T time. Consider pm_i represents the power of jth host node during t_k. Thus, the power utilization can be obtained in terms of CPU utilization at the host node. Consider, $pm_i E^{(k)}$ be the power or energy consumption of the jth host node in between the last time interval and the current time, then it can be obtained as

$$pm_i E^{(k)} = pm_{iw}(k-1) + (pm_{iw}(k-1) + (pm_{iw}(k))(t_k - t_k - 1) \tag{3.13}$$

The energy consumption for the jth host, $E(pm_j)$ can be estimated at certain host pm_j having CPU usage as $CPU_{i,Util}(k)$

$$E(pm_j) = K_j.e_j^{max} + (1 - k_j).e_j^{max}.CPU_{i,Util}(k) \tag{3.14}$$

Where K_j states the part of energy consumed when the host pm_j is in idle state; e_j^{max} states for the energy consumption of host pm_j when it being used 100%. The variable $CPU_{(i,Util)}(k)$ represents the CPU utilization by host pm_j. We have used this approach to estimate the energy consumption at certain host so as to perform placement scheduling. Similarly, the energy consumption for all host nodes, $D_E(k)$ can be obtained for a period using following equation:

$$D_E(k) = \sum_{i=1}^{m} pm_i E^{(k)} \tag{3.15}$$

In this paper, the prime objective of the proposed A-GA scheme for VM placement is to retrieve the set of mapping from VM set to the host set PM while ensuring minimal energy consumption $D_E(k)$, provided:

$$\forall_i \sum_{j=1}^{m} x_{ij-1} \tag{3.16}$$

$$\forall_j \sum_{i=1}^{n} vm_{CPU,i} X_{ij} \leq pm_{CPU,j}. \tag{3.17}$$

In this paper A-GA is compared with ACO and PABFD VM placement algorithms. The implementation of our proposed A-GA based consolidation scheme using LRR and IQR for host overload detection and MMT for VM Selection [20] is shown in Figure 13. The whole process flow for ACO and PABFD is shown in Figure 11 and Figure 12 respectively.

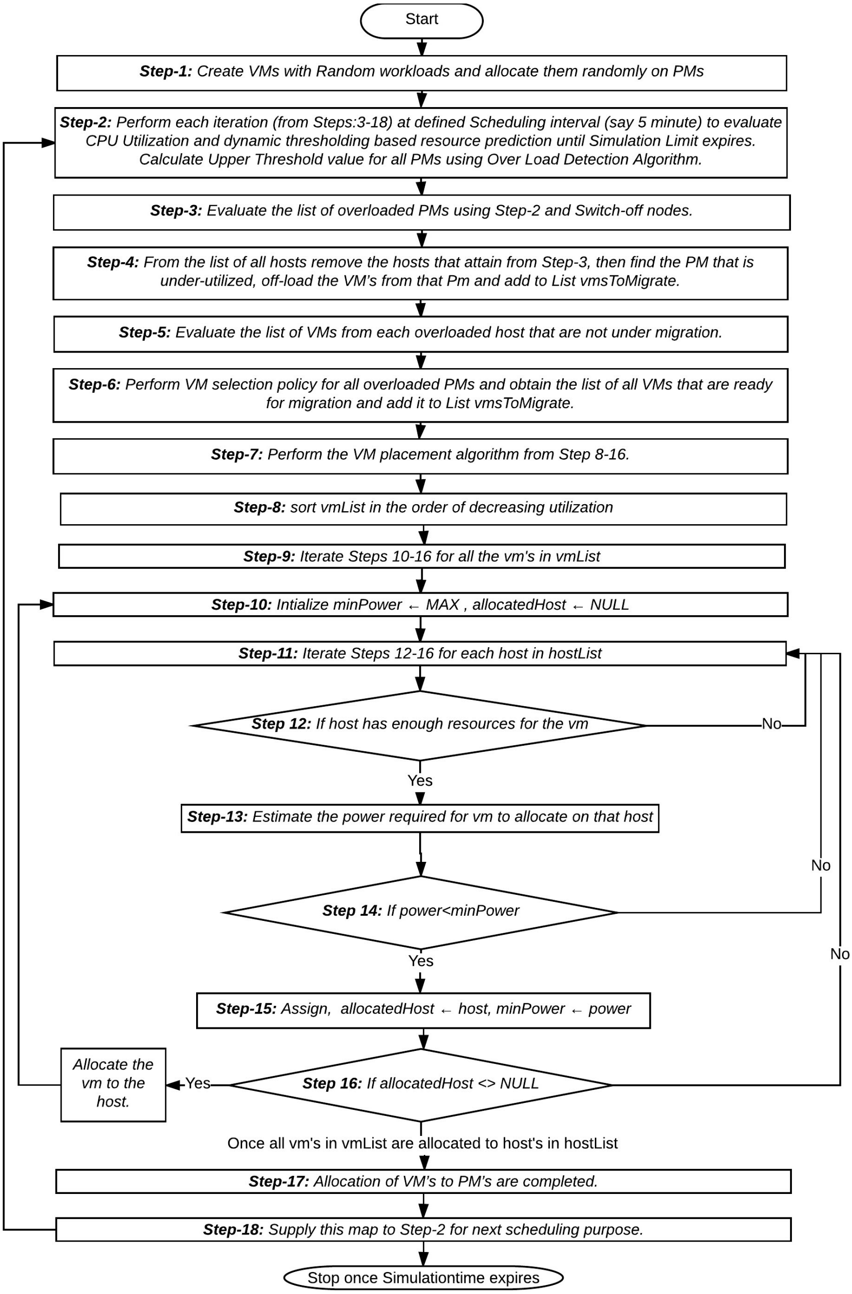
Start
Step-1: Create VMs with Random workloads and allocate them randomly on PMs
Step-2: Perform each iteration (from Steps:3-18) at defined Scheduling interval (say 5 minute) to evaluate CPU Utilization and dynamic thresholding based resource prediction until Simulation Limit expires. Calculate Upper Threshold value for all PMs using Over Load Detection Algorithm.
Step-3: Evaluate the list of overloaded PMs using Step-2 and Switch-off nodes.
Step-4: From the list of all hosts remove the hosts that attain from Step-3, then find the PM that is under-utilized, off-load the VM's from that Pm and add to List vmsToMigrate.
Step-5: Evaluate the list of VMs from each overloaded host that are not under migration.
Step-6: Perform VM selection policy for all overloaded PMs and obtain the list of all VMs that are ready for migration and add it to List vmsToMigrate.
Step-7: Perform the VM placement algorithm from Step 8-16.
Step-8: sort vmList in the order of decreasing utilization
Step-9: Iterate Steps 10-16 for all the vm's in vmList
Step-10: Intialize minPower ← MAX , allocatedHost ← NULL
Step-11: Iterate Steps 12-16 for each host in hostList
Step 12: If host has enough resources for the vm
No
Yes
Step-13: Estimate the power required for vm to allocate on that host
No
Step 14: If power<minPower
Yes
No
Step-15: Assign, allocatedHost ← host, minPower ← power
Allocate the vm to the host.
Yes
Step 16: If allocatedHost <> NULL
Once all vm's in vmList are allocated to host's in hostList
Step-17: Allocation of VM's to PM's are completed.
Step-18: Supply this map to Step-2 for next scheduling purpose.
Stop once Simulationtime expires

Figure 11: Data Flow diagram for Power-Aware Best Fit Decreasing (PABFD) based VM Consolidation Scheme

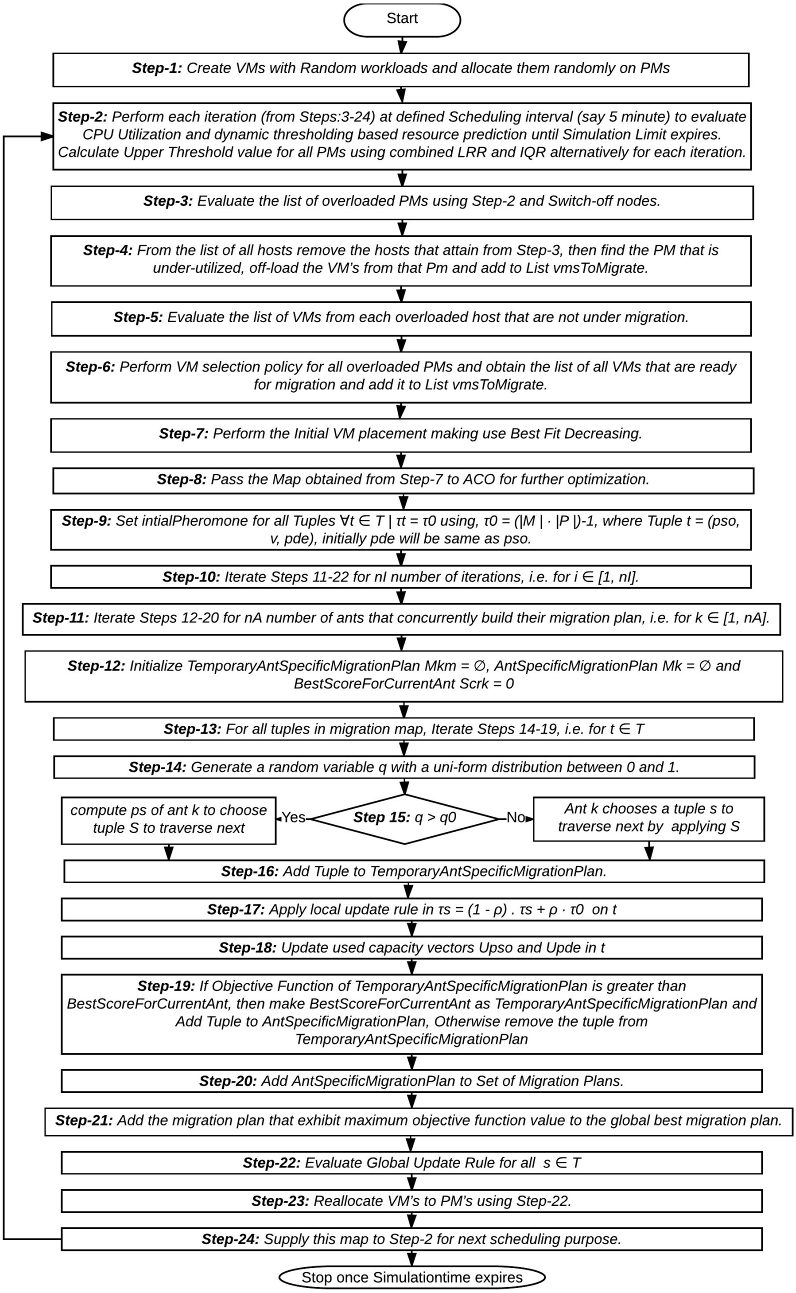
Start
Step-1: Create VMs with Random workloads and allocate them randomly on PMs
Step-2: Perform each iteration (from Steps:3-24) at defined Scheduling interval (say 5 minute) to evaluate CPU Utilization and dynamic thresholding based resource prediction until Simulation Limit expires. Calculate Upper Threshold value for all PMs using combined LRR and IQR alternatively for each iteration.
Step-3: Evaluate the list of overloaded PMs using Step-2 and Switch-off nodes.
Step-4: From the list of all hosts remove the hosts that attain from Step-3, then find the PM that is under-utilized, off-load the VM's from that Pm and add to List vmsToMigrate.
Step-5: Evaluate the list of VMs from each overloaded host that are not under migration.
Step-6: Perform VM selection policy for all overloaded PMs and obtain the list of all VMs that are ready for migration and add it to List vmsToMigrate.
Step-7: Perform the Initial VM placement making use Best Fit Decreasing.
Step-8: Pass the Map obtained from Step-7 to ACO for further optimization.
Step-9: Set intialPheromone for all Tuples ∀t ∈ T | τt = τ0 using, τ0 = (|M | · |P |)-1, where Tuple t = (pso, v, pde), initially pde will be same as pso.
Step-10: Iterate Steps 11-22 for nI number of iterations, i.e. for i ∈ [1, nI].
Step-11: Iterate Steps 12-20 for nA number of ants that concurrently build their migration plan, i.e. for k ∈ [1, nA].
Step-12: Initialize TemporaryAntSpecificMigrationPlan Mkm = ∅, AntSpecificMigrationPlan Mk = ∅ and BestScoreForCurrentAnt Scrk = 0
Step-13: For all tuples in migration map, Iterate Steps 14-19, i.e. for t ∈ T
Step-14: Generate a random variable q with a uni-form distribution between 0 and 1.
compute ps of ant k to choose tuple S to traverse next
Yes
Step 15: q > q0
No
Ant k chooses a tuple s to traverse next by applying S
Step-16: Add Tuple to TemporaryAntSpecificMigrationPlan.
Step-17: Apply local update rule in τs = (1 - ρ) . τs + ρ · τ0 on t
Step-18: Update used capacity vectors Upso and Upde in t
Step-19: If Objective Function of TemporaryAntSpecificMigrationPlan is greater than BestScoreForCurrentAnt, then make BestScoreForCurrentAnt as TemporaryAntSpecificMigrationPlan and Add Tuple to AntSpecificMigrationPlan, Otherwise remove the tuple from TemporaryAntSpecificMigrationPlan
Step-20: Add AntSpecificMigrationPlan to Set of Migration Plans.
Step-21: Add the migration plan that exhibit maximum objective function value to the global best migration plan.
Step-22: Evaluate Global Update Rule for all s ∈ T
Step-23: Reallocate VM's to PM's using Step-22.
Step-24: Supply this map to Step-2 for next scheduling purpose.
Stop once Simulationtime expires

Figure 12: Data Flow diagram for Ant Colony Optimization (ACO) based VM Consolidation Scheme

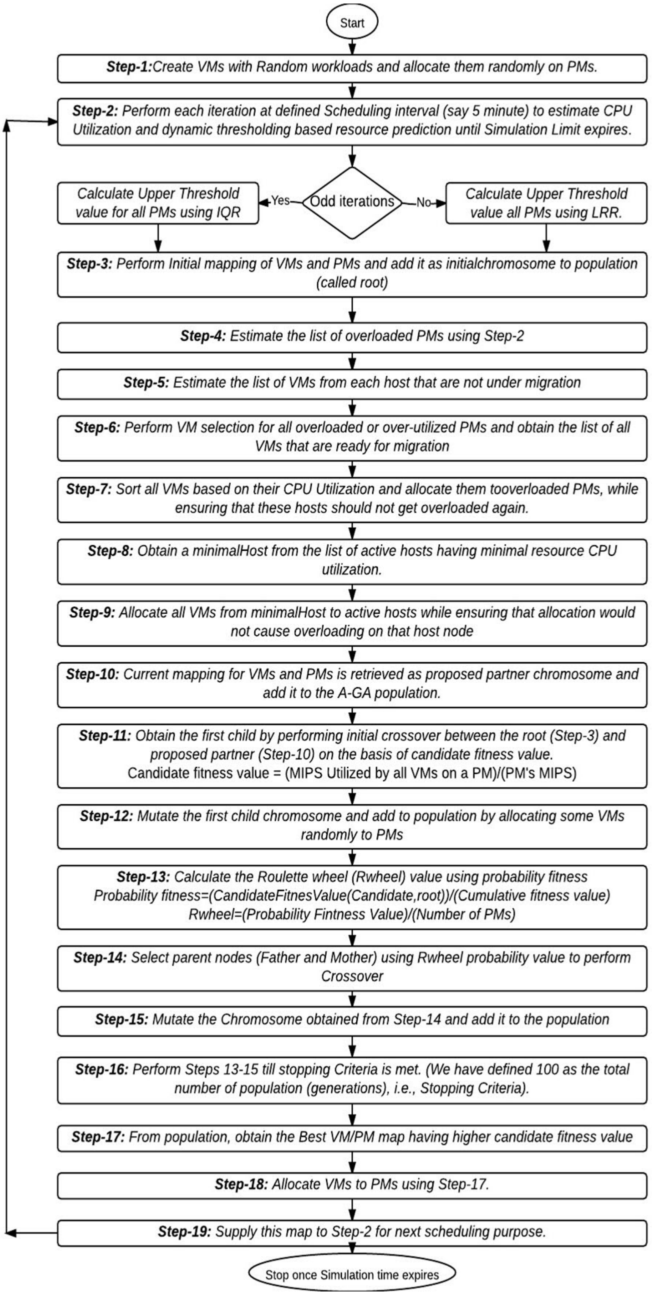
Start
Step-1:Create VMs with Random workloads and allocate them randomly on PMs.
Step-2: Perform each iteration at defined Scheduling interval (say 5 minute) to estimate CPU Utilization and dynamic thresholding based resource prediction until Simulation Limit expires.
Odd iterations
Yes
No
Calculate Upper Threshold value for all PMs using IQR
Calculate Upper Threshold value all PMs using LRR.
Step-3: Perform Initial mapping of VMs and PMs and add it as initialchromosome to population (called root)
Step-4: Estimate the list of overloaded PMs using Step-2
Step-5: Estimate the list of VMs from each host that are not under migration
Step-6: Perform VM selection for all overloaded or over-utilized PMs and obtain the list of all VMs that are ready for migration
Step-7: Sort all VMs based on their CPU Utilization and allocate them tooverloaded PMs, while ensuring that these hosts should not get overloaded again.
Step-8: Obtain a minimalHost from the list of active hosts having minimal resource CPU utilization.
Step-9: Allocate all VMs from minimalHost to active hosts while ensuring that allocation would not cause overloading on that host node
Step-10: Current mapping for VMs and PMs is retrieved as proposed partner chromosome and add it to the A-GA population.
Step-11: Obtain the first child by performing initial crossover between the root (Step-3) and proposed partner (Step-10) on the basis of candidate fitness value.
Candidate fitness value = (MIPS Utilized by all VMs on a PM)/(PM's MIPS)
Step-12: Mutate the first child chromosome and add to population by allocating some VMs randomly to PMs
Step-13: Calculate the Roulette wheel (Rwheel) value using probability fitness
Probability fitness=(CandidateFitnesValue(Candidate,root))/(Cumulative fitness value)
Rwheel=(Probability Fintness Value)/(Number of PMs)
Step-14: Select parent nodes (Father and Mother) using Rwheel probability value to perform Crossover
Step-15: Mutate the Chromosome obtained from Step-14 and add it to the population
Step-16: Perform Steps 13-15 till stopping Criteria is met. (We have defined 100 as the total number of population (generations), i.e., Stopping Criteria).
Step-17: From population, obtain the Best VM/PM map having higher candidate fitness value
Step-18: Allocate VMs to PMs using Step-17.
Step-19: Supply this map to Step-2 for next scheduling purpose.
Stop once Simulation time expires

Figure 13: Data flow diagram for proposed Adaptive Genetic Algorithm (A-GA) based VM Consolidation Scheme

Workbench Data	Number of host Nodes	Number of VM's
Random Workload	300	400, 500, 600
Random Workload	600	800,1000,1200
Random Workload	800	1000,1200,1600

Table 1: Workbench data with different number of host nodes and VM's

4. Experimental setup

In this paper, we have examined the performance of the proposed system using random cloud workload traces. To evaluate the robustness of the proposed consolidation scheme a large scale cloud infrastructure with different network size and configurations has been considered as per Table 1. For simulation we utilized CloudSim in Eclipse IDE. Intially, servers frequency is mapped to MIPS ratings with 1860 and 2660 MIPS in HP ProLiant ML110 G4 and HP ProLiant ML110 G5, respectively. Each server has been assigned 1 GB/s network bandwidth.

5. Results and discussion

The different simulation scenarios used for evaluation are mentioned in Table 2. With different network and process algorithm combination, the simulations have been done and respective performance has been analyzed in terms of the following performance parameters:

1. Number of VM Migrations (in Nos.)
2. Service Level Agreement Violation (SLAV) (%)
3. SLA time per Active Host (%)
4. SLA Performance Degradation (%)
5. Number of Host Shutdown (Nos)
6. Energy Consumption (KWh)

In this paper, we plotted graphs only for random workload with 800 hosts and VMs with 1000, 1200, 1600 (as similar results are observed for 300 and 600 hosts). As shown in Figure 14, the proposed A-GA based VM consolation yields minimal VM migration thus providing minimal downtime. Figure 15 illustrates SLAV, where the proposed A-GA provides minimal SLA violation with MMT VM selection policy.

Figure 16 represents SLA performance degradation and it clearly states proposed evolutionary A-GA with IQR and LRR as overload prediction schemes and MMT

CPU Utilization Threshold	**VM Selection**			**VM Placement**
IQR	MMT	MC	RS	Power Aware Best Fit Decreasing
LR	MMT	MC	RS	Power Aware Best Fit Decreasing
MAD	MMT	MC	RS	Power Aware Best Fit Decreasing
THR	MMT	MC	RS	Power Aware Best Fit Decreasing
LRR	MMT	MC	RS	Power Aware Best Fit Decreasing
Combined IQR and LRR	MMT	MC	RS	Ant Colony Optimization
Combined IQR and LRR	MMT	MC	RS	Adaptive Genetic Algorithm

Table 2: Implementation and simulation scenarios

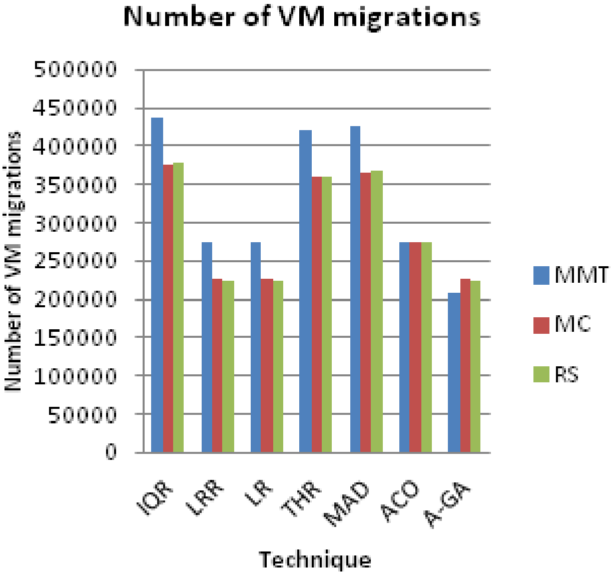

Figure 14: Number of VM migrations for 800 Hosts

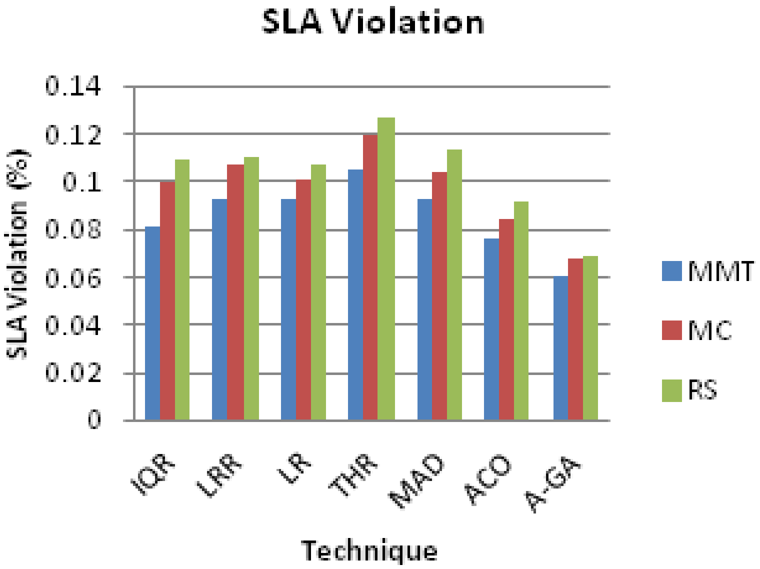

Figure 15: SLA violation for 800 Hosts

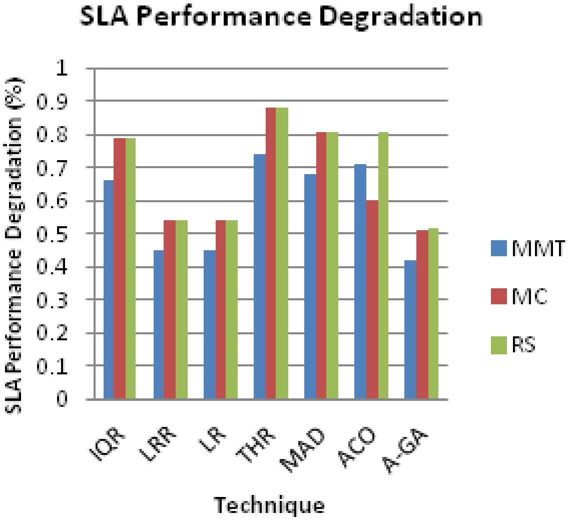

Figure 16: SLA performance degradation for 800 Hosts

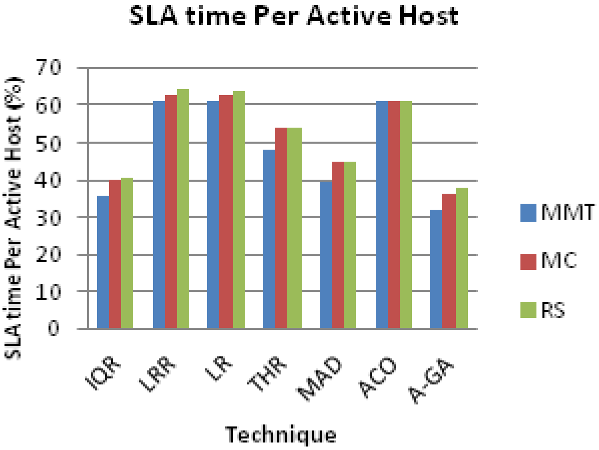

Figure 17: SLA time per active host for 800 Hosts

as VM selection policy exhibits minimal performance degradation. Figure 17, also reflects that the proposed system performed better in terms of SLA time per active host. Figure 18 affirms that maximum hosts shutdown happened with proposed system. The complete analysis states that proposed system provides complete QoS assurance with reliability.

Figure 19 clearly shows the proposed system with IQR and LRR + MMT + A-GA ensure minimal energy is consumed thus drive towards green cloud computing.From the plotted graphs it is clearly found that the proposed A-GA based scheme can provide better results as compared to the other heuristic techniques such PABFD and ACO. Taking into account among selection policies, it has been observed that MMT policy outperforms other approaches such as MC and RS.

6. Conclusion

In this paper we presented highly optimal random workbench based performance evaluation using adaptive genetic algorithm, where different complexities are considered. The execution of merging both IQR and LRR for dynamic CPU prediction schemes yielded optimal results compared with other schemes. In terms of all other paramaters MMT in combing with A-GA outperforms other approches such as MC and RS. A-GA outperforms ACO and PABFD based VM consolidation. As a reult we can quote that the proposed system can be utilized in large scale cloud data centers using random work load for green cloud computing and QoS assurance.

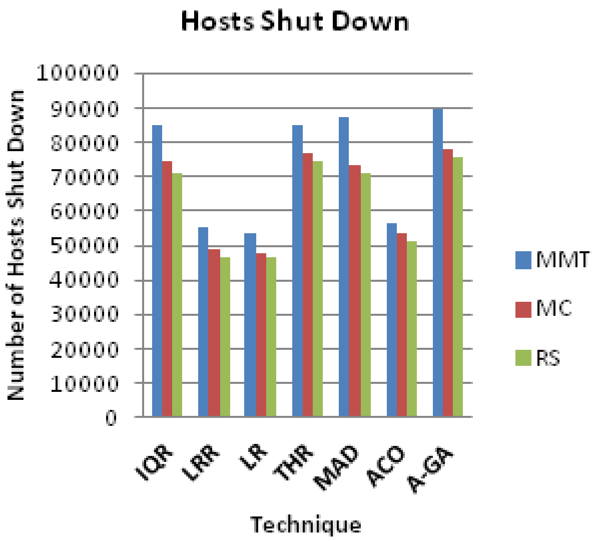

Figure 18: Number of hosts shutdown for 800 Hosts

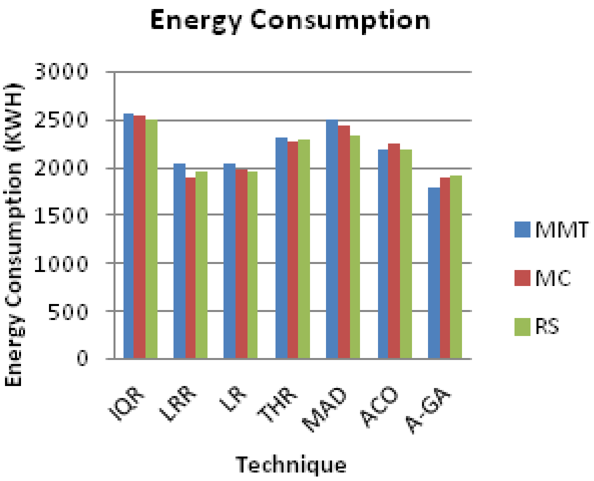

Figure 19: Energy consumption for 800 Hosts

References

[1] Y. Ajiro, A. Tanaka. Improving packing algorithms for server consolidation. - Proceedings of the International Conference for the Computer Measurement Group, San Diego, California, USA, 2007, pp. 399–407.

[2] D. Barbagallo, E. Di Nitto et al. A bio-inspired algorithm for energy optimization in a self-organizing data center. - Self-Organizing Architectures, Springer, 2010, pp. 127–151.

[3] Beloglazov, J. Abawajy et al. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. - Grid Computing and eScience, Future Generation Computer Systems, Vol 28, 2012, pp. 755–768.

[4] Beloglazov, R. Buyya. 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, Vol 24, 2012, No 13, pp. 1397–1420.

[5] Y.S. Dong,G.C.Xu et al. A distributed parallel genetic algorithm of placement strategy for virtual machines deployment on Cloud platform. -The Scientific World Journal, 2014, pp. 1–12.

[6] M. Dorigo, L. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. - IEEE Transactions on Evolutionary Computation, Vol 1, 1997, No 1, pp. 53–66.

[7] M. Dorigo, G. Di Caro et al. Ant algorithms for discrete optimization. - Artificial Life, Vol 5, 1999, No 2, pp. 137–172.

[8] A. Esnault, E. Feller et al. Energy-aware distributed ant colony based virtual machine consolidation in IaaS Clouds bibliographic study. - Informatics Mathematics (INRIA), 2012, pp. 1–13.

[9] F. Farahnakian, P. Liljeberg et al. Linear regression based CPU usage prediction algorithm for live migration of virtual machines in data centers. - Software Engineering and Advanced Applications, 39th Euromicro Conference, Santander, Spain, 2013, pp. 357–364.

[10] F. Farahnakian, T. Pahikkala et al. Energy aware consolidation algorithm based on K-nearest neighbor regression for Cloud data centers. - 6th IEEE/ACM International Conference on Utility and Cloud Computing, Dresden, Germany, 2013.

[11] E. Feller, C. Morin et al. A case for fully decentralized dynamic VM consolidation in Clouds. -Cloud Computing Technology and Science, 4th IEEE International Conference, Taipei, Taiwan, 2012, pp. 26–33.

[12] M.H. Ferdaus, M. Murshed et al. Virtual machine consolidation in Cloud data centers using ACO metaheuristic. - Euro-Par 2014 Parallel Processing, 20th International conference, Porto, Portugal, 2014, pp. 306–317.

[13] M. Harman, K. Lakhotia et al. Cloud engineering is search based software engineering too. - Journal of Systems and Software, Vol 86, 2013, No 9, pp. 2225–2241.

[14] Hongwei Chen, Lei Xiong et al. Cloud task scheduling simulation via improved ant colony optimization algorithm. - Journal of Convergence Information Technology, 2013.

[15] B. Madhusudhan, K.C. Sekaran. A genetic algorithm approach for virtual machine placement in Cloud. - Proceedings of International conference on Emerging research in computing, information, communication and applications, 2013, pp. 115–122.

[16] M. Marzolla, O. Babaoglu et al. Server consolidation in Clouds through gossiping. -World of Wireless, Mobile and Multimedia Networks, 12th IEEE International Symposium, Lucca, Italy, 2011, pp. 1–6.

[17] Murtazaev, S. Oh.Sercon: Server consolidation algorithm using live migration of virtual machines for green computing. - IETE Technical Review, Vol 28, 2011, No 3, pp. 212–231.

[18] Perla Ravi Theja, SK. Khadar Babu. An Evolutionary Computing based Energy Efficient VM Consolidation Scheme for Optimal Resource Utilization and QoS Assurance. - Indian Journal of Science and Technology, 77179, Vol 8, 2015, No 26, pp. 1–11.

[19] Perla Ravi Theja, Dr. SK. Khadar Babu. An Adaptive Genetic Algorithm based Robust QoS Oriented Green Computing Scheme for VM Consolidation in Large Scale Cloud Infrastructures. - Indian Journal of Science and Technology, 79175, Vol 8, 2015, No 27, pp. 1–13.

[20] Perla Ravi Theja, SK. Khadar Babu. Evolutionary computing based QoS oriented energy efficient VM consolidation scheme for large scale cloud data centers. - Cybernetics and Information Technologies, Bulgarian Academy of sciences, Vol. 16, No. 2, pp. 97–112.

[21] Pierre Delforge. America's Data Centers Consuming and Wasting Growing Amounts of Energy. -Available at : https://www.nrdc.org/resources/americas-data-centers-consuming-and-wasting-growing-amounts-energy , February,06, 2015.

[22] M. Tang, S. Pan, A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers. - Neural Processing Letters, Vol 41, 2015, No 2, pp.211–221.

[23] W. Vogels.Beyond server consolidation. - ACM Queue, Vol , 2008, No 1, pp. 20–26.

[24] M. Wang, X. Meng et al. Consolidating virtual machines with dynamic bandwidth demand in data centers. - 30th IEEE International conference on computer communications, Shanghai, China, 2011, pp. 71–75.

[25] T. Wood, P. Shenoyet al. Sandpiper: Black-box and gray-box resource management for virtual machines. - Computer Networks, Vol 53, 2009, pp. 2923–2938.

[26] H. Zhong, K. Tao et al. An approach to optimized resource scheduling algorithm for open-source Cloud systems. - China Grid Conference (China Grid), 2010 Fifth Annual, Guangzhou, China, 2010 pp. 124–129.