
Suresh B. Rathod*, V. Krishna Reddy

Abstract—In decentralized cloud architecture, the host’s configured with an autonomous local resource manager (ALRM) which takes decisions for Virtual Machine (VM) migration if it is over utilized. The ALRM takes decision for migrating its one of the VM to other peer host, by considering the peer host’s utilizations received after fixed interval. This autonomous decision making results in same host identification by multiple hosts. The VM placement might results in the identified server to get over utilized and it might initiate the process of VM migration. During migration the VM and its content migrated to the identified host in plaintext form. This involves the user credentials and VM’s kernel state information. Hence fault tolerance aware secure VM migration for decentralized cloud computing is introduced which avoids the over utilization of the identified server by considering its future CPU utilization, avoids the same host identification by hybrid decentralized decision making and it also ensures the VM data remain protected during migration. If failure in the decision making model the fault tolerance mechanism is introduced that helps to maintain system up for longer time. Experimental results reveals that the proposed solution helps in providing security to VM’s data during VM migration and avoids same destination host selection during VM placement.

Keywords—Cloud computing (CC), Virtual Machine (VM), Controlling Host (CH), Host controller (HC).

I. INTRODUCTION

In recent years, cloud computing gaining more popularity because of its ability to provide virtualized resources. In virtualized environment VM acts as the core component, it provides uninterruptable services to end user. VM runs on top of the hypervisor and consumes underlying host’s resource. Each VM differs with other VMs in the resource it consumes, including processor architecture, operating system (OS), memory type, network bandwidth and the tasks it is executing. This results in each host in DC would have multiple VM’s instances running parallel with different job completion time [29]. As per NIST standard [1], virtualization addresses varying resources requirement raised by the VM’s applications. The varying resource requirement by applications running on the VM gets fulfilled by the underlying physical host’s resource pool. These virtualized resources assigned to the VM by the underlying hosts resource manager using partitioning, isolation, and encapsulation [1][2]. If the underlying host is unable to fulfill the resources requirement raised by the VM or the VM is consuming more resources than allocated then the underlying virtual machine manger initiates the process of VM migration from current host to the identified host. There are hypervisors like KVM, XEN, Hyper-v and ESXi provides utility functions (API) that facilitates underlying host’s resource management[20].

VM migration is either the static or live migration [23]. In static VM migration the VM stops its execution at origination host, VMM migrates its resources to other host. After resource migration the VM resumes its execution at migrated host. This migration type requires manual interruption. In live VM migration, the running VM instance paused at source host and resumes its execution to the destination. In live VM migration, the VM’s memory, its processor state, and the network details migrated to the destination host in rounds[23]. The live migration done either in pre-copy or post-copy approach [23]. In post-copy migration, the VM’s memory migrated to the destination host after the VM’s processor state migrated [23]. Pre-copy migration, the VM’s pages migrated to destination host, at last the processor state migrated [23].

Various authors have discussed VM migration decision making framework considering centralized or decentralized cloud architecture [29]. The cloud vendors like Google, Amazon, HP, and IBM provides cloud services to end user adopting either architecture. In decentralized cloud architectures, each host configured with ALRM which runs independent and also takes decision for VM migration by its own. To do this, ALRM refers the neighbour host detail received at fixed interval. This own decision making by each host leads to the problem of same host identification by multiple host and over utilization of the identified host. These results in destination host might initiate VM migration.

Fig.1: Decentralized cloud computing decision making framework

---

Manuscript received February 01, 2019

Suresh B. Rathod*, Research Scholar, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh, India. (e-mail: sureshrathod1@gmail.com)

V. Krishna Reddy, Professor, Department of computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh, India.

Retrieval Number: E11450275SI19/19©BEIESP

Published By:
Blue Eyes Intelligence Engineering & Sciences Publication
Cloud users do access and deploys their applications on cloud using cloud services. Cloud user can access and managed these cloud services from remote location. Each application has varying resource requirement, this leads to extra bourdon on the underlying host. This varying resource requirement from each application running on VM causes underlying host to be either unutilized or over utilized. The decentralized cloud architectures is shown in Fig.1, where in each host share its detail to other peer host after fixed interval. Using the peer information from the peer hosts, the ALRM running at each host takes decision for VM migration, if it finds it is over utilized due to varying resource requirement by the running VM. It migrates its running VM to other peer host using the peer information received at fixed interval. The peer in decentralized host never shares the host identification done with other peer and its future utilization after migrating its VM. This results in chances of identification of same host by multiple peer hosts. This causes sometimes the over utilization of the identified host. This requires incorporation of new decentralized framework, where the resource manager considers host’s current and future CPU utilization during the VM migration and ensures secure VM migration to other hosts. This framework also ensures the decision making capability among the peer remains up by introducing the fault tolerance.

Remaining portion of this paper organized as follows: section 2 describes the related work, section 3 discusses proposed system, section 4 discusses the results of the proposed system and at last the conclusion.

II. RELATED WORK

VM’s migrated to other host to avoid over utilization of the host that is caused due to excessive resource requirement by the running VM or over utilization caused due to VM placement. Various authors have discussed techniques for VM decision making.

Energy based VM placement discussed by the author in [6], here, the decision for VM migration from the host done considering penalty cost and energy consumption by the host. The solution proposed by the authors suffers with poor performance for the case when the energy cost and penalty cost increases.

Author in [7], discusses the CPU utilization based distributed load balancing on hypercube based model. Here, the individual host does takes decision for VM placement without considering the destination hosts future CPU utilization.

Optimum dynamic VM placement policy proposed by the authors in [8] works on CPU consumption by the host, authors in their work discussed maximum processing power (MPP) and random host’s selection (RS) techniques for VM migration.

In [9] the author have proposed Hierarchical Decentralized Dynamic VM Consolidation Framework for VM migration, wherein they discussed how the global controller takes decision for VM migration by considering hosts future CPU utilization. The author in their work proposed the solution for VM placement using Ant Colony Optimization (ACO) technique. Again the originating host doesn’t share the ant information to other, every host on overload initiates the ant to find the host with minimum utilization. There are chances to identify the same host by multiple host and initiating the migration.

Distributed load balancing using CPU utilization proposed by the author in [10] considered hypercube based VM migration. Randomized probabilistic model proposed by the author in [10], have discussed finding host’s pair formation randomly and initiating VM migration in the selected host pair. The approach might skip the over loaded host during host selection, if it has more overloaded host.

Correlation based VM placement on centralized cloud architecture proposed by [12]. The approach might take more time to process the information, if the data size is large enough.

Muti target based VM placement using genetics algorithm proposed by the author in [14] considered SLA violation and CPU utilization as the parameter for VM migration decision making on centralized cloud architecture. The approach might lead to centralized failure.

In [15], authors have proposed Reinforcement Learning based VM placement wherein the authors have discussed how the centralized host learns VM deployment and puts host in sleep mode or in active mode considering the past traces.

The author [25] in his work proposed the decentralized VM migration on decentralized environment. The mechanism proposed in [25] deals with threshold based VM selection policy using upper and lower threshold limit. VM migrated to other server if its utilization reached to upper limit. There are chances that due to VM migration the selected host might be over utilized and might reinitiate VM migration.

The next section discusses the proposed decentralized predictive VM migration in decentralized cloud environment.

III. DECENTRALIZED SECURE PREDICTIVE VM PLACEMENT

This section discusses the architectural component followed by proposed predictive secure VM placement approach for the decentralized cloud environment.

3.1. The Architectural components

Here, the proposed architectural components followed by the predictive VM placement is discussed. The proposed architecture formed by incorporating distributed features like multi-tenant, distributed storage, parallel processing and multithreading [20].

Each host in the proposed architecture configured with the component shows in Fig.3.1. Here, the hosts are categorized as Controller Host (CH) and Host Controller (HC) based on the type of job they may perform. Host will termed as CH, if it does the task of providing services to end world and sharing information with HC at fixed interval. The host is termed as the HC, if it does the task of decision making and provides services to end world.
HC Resource Monitor (HCRM): This component activated when the CH acts as HC and does the task of decision making. It performs following tasks.
1. Collecting and storing peer hosts detail in current and past utilization table.
2. Providing host information to the Virtual Host Manager (VHM) as and when required.

Local Resource Monitor (LRM): This component interacts with the underlying hypervisor and does following tasks.
1. Collects underlying host detail
2. Share underlying host detail with HCRM after fixed interval.

Virtual Host Manager (VHM): Unlike HCRM, this performs following tasks.
1. Identifying source and destination during VM migration.
2. Predicting destination CH’s future CPU utilization.
3. Finding upper threshold limit of the participating host.
4. Sharing next HC address with all CH.

HC initiate the procedures for collecting CH detail, managing remote VM’s instances, and the identification of new HC. These threads wakes up after fixed delay set by the administrator.

\[ H_u = \sum_{i=0}^{VM_i} \] (3.1)

The CH retrieves underlying CH’s CPU utilization using Eq.(3.1) and shares with the HC after fixed interval.

Here \( H_u \) is the host utilization of server \( u \). It is the sum of all \( VM_i \) running on the host \( u \) at time interval \( t \).

\[ MAD = \frac{\sum_{i=1}^{n} y_i^2}{n} \] (3.2)

VHM calls HCRM to start retrieving active CH connections using GETCONNECTION. VHM finds source and destination CH address which has minimum and maximum CPU utilization. To do so, it uses GETMAX and GETMIN to find the minimum and maximum CH address. Upon identifying hosts, the VHM predicts the future CPU utilization of each CH and stores in FHOST(future CPU utilization).

In this proposed architecture, the CH has the heterogeneous hardware configuration, static threshold might not be the good solution. CH’s upper threshold computed using equation (3.2). Here, Mean Absolute Deviation MAD [9] used to find CH’s upper threshold.(Dynamic threshold) computed using Eq. (3.3).

Here, the \( y_i \) represent real CH’s utilization and \( n \) represent a number of observations till now and \( \hat{y} \) represent fitted value at time \( t[20] \).

UpperThreshold = 1-MAD (3.3)

After identifying CH with maximum CPU utilization, VHM searches VM with minimum CPU utilization and marks such VM for migration.

![Fig. 2: Proposed decentralized hybrid host component diagram.](image-url)
The identified VM said to be successful if it satisfies the following conditions.

- The destination CH has its FHOST lesser than the upper threshold.
- The CH’s upper threshold is lesser than or equal to 0.9.

The VM placement is unsuccessful if it satisfies following conditions.

- CH’s current utilization of the destination CH is greater than 0.9.
- The FHOST value of destination CH is lesser than upper threshold.

To deal with failure case, VHM starts searching new CH that has CPU utilization lesser than the current identified destination CH. The new CH selected if it has sufficient resources. The algorithm for FINDNEXT is shown in algorithm 2.

### Algorithm 2 FINDNEXT

1. procedure FINDNEXT(destAddress, srcAddress, HOSTLIST)
2. for each host i in HOSTLIST do
3. if HOSTLIST[i] ≤ HOSTLIST[i + 1] then
4. temp = HOSTLIST[i] + [HOSTLIST[i] + HOSTLIST[i + 1] + temp]
5. end if
6. end for
7. return HOSTLIST

The future utilization of the CH computed using Doubles Exponential Smoothing (DES) [16]. Eq.(3.6) shows CH’s future CPU utilization and smoothed value calculations.

\[ s_t = \alpha y_t + (1 - \alpha)(s_{t-1} + b_{t-1}), 0 \leq \alpha \leq 1 \]  \hspace{1cm} (3.4)

\[ b_t = \gamma(s_t - s_{t-1}) + (1 - \gamma)b_{t-1}, 0 \leq \gamma \leq 1 \]  \hspace{1cm} (3.5)

\[ f_{t+m} = s_t + mb_t \]  \hspace{1cm} (3.6)

Here \( s_t \) represents CH’s smooth values at time \( t \) and \( y_t \)represents observed values over period \( t \) [20]. \( b_t \) represent trend factor over time period \( t \) values for the previous period \( t-1 \). This \( f_{t+m} \)called the smoothing function [20].

### 3.3 Decentralized secure peer to peer VM placement

Umesh Deshpande and Kate Keahey(2015) has discussed NAS based VM migration. In their work they discussed during live VM migration VM’s processor state, allocated RAM content and the data stream linked with each running task migrated to the destination host. The data stream might contain sensitive and confidential user data. In order to protect data during migration, the secure tunnel need to be established in the CH’s. Tunnel formation in Linux/Unix platform achieved by SSH.

Here, the every CH configured with other CH’s public key before they registers with central host. Fig.3.2 shows the SSH setup in CHs. Following are the steps to setup the SSH between CH before they start sharing information. From figure 4. The VM migration is initiated from the CH1 to CH2. Before VM is migrated it initiates the following procedure.
HCRM after set interval initiates call to find CH having minimum CPU utilization and the maximum CPU utilization at current instance of time.

The VHM upon receiving the CH address starts using past utilization table and initiates thread to find the future utilization of the CH having minimum CPU utilization at current instance.

From Table 1, CH with address 10.0.0.3 has maximum CPU utilization compared with 10.0.0.2 and 10.0.0.4. CH with 10.0.0.3 marked as destination CH and 10.0.0.2 marked as origination CH.

The VHM initiates DPPVP and finds future utilization of the 10.0.0.3 and 10.0.0.4. It also initiates the process to find the upper threshold limit of CH 10.0.0.3 and the CH 10.0.0.4.

The VHM compares upper threshold limit for the CH 10.0.0.3 with 0.9 and performs following checks.

- If the upper limit is greater than 0.9 it will set it to 0.9.
- If the upper limit is less than 0.9 and has its future utilization less than 0.9 then the process of VM migration initiated.

If the upper limit is less than future CPU utilization then new CH identified and above two checks performed.

Before VM migration initiated, the VHM initiates a call to find the VM address that has minimum CPU utilization and marked such VM for migration. Once VM identified from the source CH 10.0.0.2 the secure channel established between CH 10.0.0.2 and 10.0.0.4. The selected VM migrated to host 10.0.0.3.

**Table 2: Past utilization table at HC**

<table>
<thead>
<tr>
<th>SERVER</th>
<th>CPU utilization</th>
<th>NO. VM</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.3</td>
<td>0.0804</td>
<td>2</td>
<td>FALSE</td>
</tr>
<tr>
<td>10.0.0.2</td>
<td>0.1227</td>
<td>2</td>
<td>FALSE</td>
</tr>
<tr>
<td>10.0.0.4</td>
<td>0.0811</td>
<td>2</td>
<td>TRUE</td>
</tr>
<tr>
<td>10.0.0.3</td>
<td>0.08</td>
<td>2</td>
<td>FALSE</td>
</tr>
<tr>
<td>10.0.0.2</td>
<td>0.0756</td>
<td>1</td>
<td>FALSE</td>
</tr>
<tr>
<td>10.0.0.4</td>
<td>0.0936</td>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>10.0.0.3</td>
<td>0.1014</td>
<td>2</td>
<td>FALSE</td>
</tr>
<tr>
<td>10.0.0.2</td>
<td>0.0756</td>
<td>1</td>
<td>FALSE</td>
</tr>
<tr>
<td>10.0.0.4</td>
<td>0.0936</td>
<td>3</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

The proposed approach does task to maintain the CHs utilization less than its maximum upper threshold limit. The CH said to be in normal state, if it’s upper and current utilization lesser than 0.7. CH said to be over utilized if it’s current and upper threshold CPU utilization greater than 0.7.

![Fig.4: CH’s utilization of server](image)

Fig. 4 shows CH’s utilization and Fig.5 shows the Number of VM running on each CH. From Fig.4.1 and Fig.5 it is observed that the CH’s utilization gets reduced due to migrating the VM from. The CH with address 10.0.0.2 has maximum CPU utilization and 10.0.0.4 has minimum utilization, so the VM from 10.0.0.2 migrated to the 10.0.0.4.

![Fig. 5: No of VM on CH’s.](image)

From Fig.5 it is observed that the VM after initial migration there is no change in number of VM running on the CH 10.0.0.3. The CPU utilization CH 10.0.0.4 and 10.0.0.2 is changing so there is continuous VM migration in them. The VM is migrated to them as they have their future utilization lesser than 0.9.

Comparing proposed system with existing systems, the proposed system considers destination host’s utilization. The ALRM activated only when the CH acts as the HC. This results in avoiding same CH identification by multiple host during VM placement. The proposed architecture has the capability to work in failure. To deal with HC failure a reserved host is added which helps to transfer the decision making capability among the peer hosts in proposed architecture.

To do so, after fixed interval set by the administrator, if the peer server does not receive any response from the HC after fixed interval, each server starts connection to the reserved host. From the Fig.3.1, if the CH, CH4 goes down, all the remaining CHs start connecting to the reserved host address configured with them. Upon receiving CH’s details, the reserved host finds the next HC address and shares this HC address to all CH’s. After broadcasting, the reserved host updates the new HC address to the HC list for further reference.

Fig.6 shows the HC output window after applying the fault tolerance.

V. CONCLUSION

The host categorization into CH and HC avoids same host identification by multiple hosts. The proposed approach secures VM’s data by establishing secure channel during VM migration process. Proposed approach restricts message exchange in HC and CHs leads to preserving message exchange in every CH, this results in less bandwidth consumption for message exchange. Fault tolerance mechanism incorporation helps in avoiding HC’s failure. Prediction by DES smoothing ensures destination host never get over utilized due to VM placement and avoid unnecessary VM migration initiation due to VM placement at destination host.

ACKNOWLEDGEMENT

We would like thanks to the Doctoral committee and all the faculty members at Koneru Lakshmaiah Education Foundation and Sinhgad Academy of Engineering to support this work.

REFERENCES


