ISSN 2395-1621

Virtual Machine Replacement Using Optimal Placement Strategy Searching

[#1]Shital Ghorpade, [#2]Chaitali Jagtap, [#3]Shamali Jagtap, [#4]Monika Manjare, [#5]Sneha Tirth.

[#1234]Under Graduate, Department of Information Technology,
[#5]Assistant Professor, Department of Information Technology,

Trinity College of Engineering and Research, Pune-48

ABSTRACT

Today's IT services are moving to cloud computing environment in order to process client request and provide services effectively. In this case, reliability is a major factor to reduce the load in storage and network resource. In traditional method, it uses fault tolerance, replication of VM and storing checkpoint image in a neighboring server. Replication increases cost for a large system and checkpoint image inaccessibility occurs if any node crashes. It can be rectified by distributed storage of checkpoint across every server. This persistent storage can be made using a heuristic algorithm – an optimization problem. In this paper, survey is collected to measure the performance of various types of enhancing reliability.

Keywords: Cloud computing, Fault tolerance, Virtual Machine.

ARTICLE INFO

Article History

Received: 2nd May 2017

Received in revised form :

2nd May 2017

Accepted: 4th May 2017

Published online :

9th May 2017

I. INTRODUCTION

Cloud computing has evolved as an important and popular computing model . Similar to public utility services, computing resources in a cloud computing environment can be provisioned in an on-demand manner, and can be purchased via a pay-as-you-go model .This obviates the need to costly over-provision on-premise computing resources to accommodate peak demand .Thus, deploying services into the cloud has become a growing trend.

Reliability is an important aspect of Quality of Service(QoS) . With many virtual machines (VMs) running in a cloud data centre, it is difficult to ensure all the VMs always perform satisfactorily . In reality, many cloud services failed to fulfil their reliability assurance commitment due to VM failures . It is imperative to enhance the reliability of VM-based services in a cloud computing environment .Many solutions have been proposed to address service reliability issues. Fault removal, fault prevention, fault fore-casting, and fault tolerance are four basic reliability enhancement techniques. The first three of them attempt to identify and remove faults that occur in the system with he goal of preventing impact-making faults. This goal is unrealistic for a complex computing system like a cloud computing environment in production, in which VM failure is inevitable.

Enhancing the reliability of cloud services is an important aspect of cloud computing and has received considerable attention from the research community. The complex cloud computing environment poses particular challenges to researchers. A variety of service reliability enhancement approaches have been proposed to address related issues.

II. LITERATURE SURVEY

Computing delivered as a utility can be defined as "on demand delivery ofinfrastructure, applications, and business processes in a security-rich, shared,scalable, and based computer environment over the Internet for a fee. Services provided by some vendors have a less performance to provide a Virtual Machines[1]. Delivers a comprehensive fault tolerance solution to user's applications by combining selected fault tolerance mechanisms. As certain the properties of a fault tolerance solution by means of runtime monitoring. Based on the proposed approach, we design a framework that easily integrates with the existing Cloud infrastructure and facilitates a third party in offering fault tolerance as a service. Check point is the points from where backward error recovery is done in case of complete failure of a system. This scheme provides an automatic forward recovery. If a node fail to produce output or produce output after time overrun the system will not fail. It will continue to operate with remaining nodes. it can't be replaced. [2]. The time to make a checkpoint mainly depends on two different aspects: creating the checkpoint and uploading it to the

checkpoint storage. This section evaluates the cost of making a checkpoint depending on the size of the memory of a VM, the size of user disk and its usage This can also lead to computation time wasting when the crash occurs before finishing the task execution [3]. The experimental results show that by tolerating faults of a small part of the most significant components, the reliability of cloud applications can be greatly improved. The demand for highly reliable cloud applications is becoming unprecedentedly strong.[4]

III. PROPOSED SYSTEM

Check pointing is a widely used basic fault tolerance mechanism that functions by periodically saving the execution state of a VM as an image file. However, datacenter save limited network resources and may readily become congested when a huge number of checkpoint image file share transferred. Attempting to avoid this problem, presented a theoretical delta-checkpoint approach in which the base system only needs to be saved once the first checkpoint completes and subsequent checkpoint images only contain the incrementally modified pages. A theoretical delta-checkpoint approach was implemented**.**

Replication is another type of reliability assurance mechanism. Replication is based on the exploitation of redundancy. One-to-one and one-to-many standbys are two well known mechanisms. tried to map each primary VM to a backup VM. A primary VM and its map-ping backup node form a survivable group. A task can be completed in time if at least one VM in the survivable group works well. The work takes the bandwidth reservation into consideration when solving the mapping problem. In this regard, an optimal mechanism that maps a survivable group to the physical data centre was proposed.

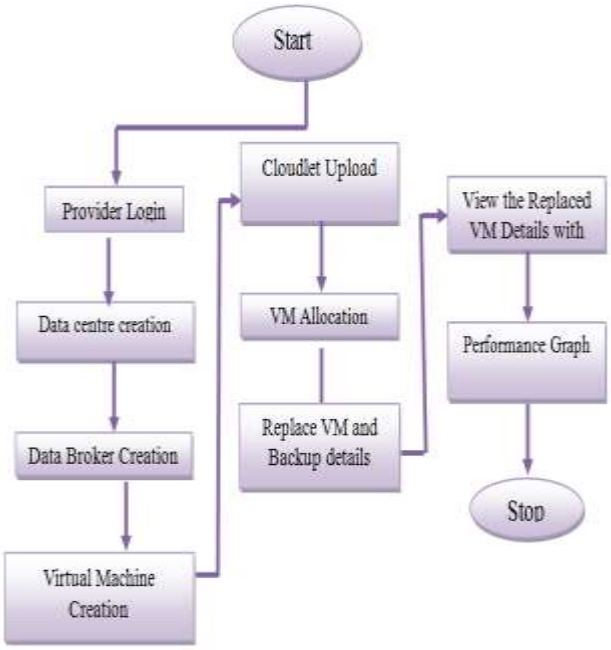

Fig 1. Architecture

Algorithm 1: Host Server Selection

Input: the number of needed primary VMs m,the number of needed backup VMs k
Output: list of interesting host servers *servers*

sort all subnets by the number of available host servers and $subnet = subnets->head$;
if $(subnet->freeServerSize) \geq (m+k)$ **then**
  goto final2;
end
sort *pods* by the number of available host servers;
$pod = pods.head$;
if $(pod.freeServerSize) \geq (m+k)$ **then**
  $next$ = Succ(pod);
  while $(next.freeServerSize) \geq (m+k)$ **do**
    $pod = next$ and $next$ =Succ(pod);
  end
  add all subnets in *pod* to *subnets* and goto final1;
end
else
  add all available host servers in *pod* to *servers*;
  while $size(servers) < (m+k)$ **do**
    pod = Succ(pod);
    if $size(pod.freeServerSize)+size(servers) > (m+k)$ **then**
      add all subnets in *pod* to *subnets*;
      goto final1;
    end
    else
      add all available host servers in *pod* to *servers*;
    end
  end
end
final1:
sort *subnets* by the number of available host servers;
$subnet = subnets.head$;
final2:
if $(subnet.freeServerSize)+ size(servers) \geq (m+k)$ **then**
  goto final 3;
end
else
  add all available host servers in *subnet* to *servers*;
  while $size(servers) < (m+k)$ **do**
    $subnet$ = Succ($subnet$);
    if $(subnet.freeServerSize)+size(servers) \geq (m+k)$ **then**
      goto final3;
    end
    else
      add all available servers in *subnet* to *servers*;
      return *servers*;
    end
  end
end
final3:
$next$ = Succ($subnet$);
while $(next.freeServerSize)+ size(servers) \geq (m+k)$ **do**
  $subnet = next$ and $next$ =Succ($subnet$);
end
select $(m+k)$- size(*servers*) available servers from *subnet*, and assigned them to *servers*;
return *servers*;

Algorithm 2: Virtual Machine Placement on Specific Servers

Input: interesting host servers *servers*, the number of backup servers *k*
Output: int *strategy*[]

```
Obtain all the "good" subnets that at least have one "good" host server;
Store the "good" host server number of each interesting subnet to vector subnets[];
Sort subnets[] by the number of "good" host server desc;
int strategy[size(subnets)];
int optimal[size(subnets)];
int mincost = ∞;
optimal placement strategy searching(subnets,strategy,optimal,k,mincost,1);
return strategy;
```

Algorithm 3: Optimal Placement Strategy Searching

Input: *subnets*, *strategy*, *optimal*, the number of un-placed backup VMs *rest*, *mincost*, nested level *i*
Output: The placement strategy

```
if rest == 0 then
    strategy[i] = rest;
    Compute cost of current strategy;
    if currentcost ≤ mincost then
        minCost=currentcost;
        copy(strategy[], optimal[]);
    end
    return;
end
if i == size(subnets) then
    if rest > min(strategy[i-1], ceil(subnet[i]/2)) then
        return;
    end
    strategy[i] = rest;
    Compute cost of current strategy;
    if currentcost ≤ mincost then
        mincost=currentcost;
        copy(strategy[], optimal[]);
    end
    return;
end
n = min(strategy[i-1], ceil(subnets[i]/2), rest);
while n ≥ 0 do
    strategy[i]=n;
    optimal placement strategy searching(subnets,strategy,optimal,k-n,cost);
    n–;
end
return;
```

Algorithm 4: Recovery Strategy Decision

Input: Set of all failed host servers VM_F, set of all backups VM_B, $w(vm_f, vm_b)$ for each $vm_f \in VM_F$ and $vm_b \in VM_B$
Output: Map *maps* between each failed host server and its backup

```
add all subnets that contains at least one backup VM to subnets ;
for each subnet subnet_i in subnets do
    add VM_F ∩ VM(subnet_i) to list srcVM;
    add VM_B ∩ VM(subnet_i) to list dstVM;
    sort srcVM_f by data size;
    while srcVM is not ∅ and dstVM is not ∅ do
        key = srcVM-> head;
        value = dstVM-> head;
        add < key ,value > to maps;
        remove key from PM_F and srcVM ;
        remove value from PM_B and dstVM ;
    end
end
if VM_F is not ∅ then
    for each pod pod_i in pods do
        clear srcVM and add VM_F ∩ VM(pod_i) to list srcVM;
        clear dstVM add VM_B ∩ VM(pod_i) to list dstVM;
        while srcVM is not ∅ and dstVM is not ∅ do
            key = srcVM-> head;
            value = dstVM-> head;
            add < key ,value > to maps;
            remove key from PM_F and srcVM ;
            remove value from PM_B and dstVM ;
        end
    end
end
if VM_F is not ∅ then
    while VM_F is not ∅ do
        key = VM_F -> head;
        value = VM_B -> head;
        add < key ,value > to maps;
        remove key from PM_F;
        remove value from PM_B;
    end
end
return maps;
```

MODULES

- Login Page
- Datacenter Creation
- Data broker Creation
- Virtual Machine Creation
- Cloudlet Uploading
- Virtual Machine Allocation
- Reallocate VM and Backups

MODULES DESCRIPTION

Login Page

If user login into the valid username and password means it may navigate into the next page. Otherwise it show an alert message and these details are fetch from the database tables.

Fig2:Provider Login

Datacenter Creation

Create a datacenter name, no of machines, host id, Ram and bandwidth using Cloud sim tool. In this module user enter the number of datacenters. It may create a corresponding number of datacenters. These details are stored in the database and fetch from the database and displayed.

Fig3:Datacenter Creation

Data Broker Creation

Create a broker id and broker name using Cloud sim tool. In this module user enter the number of data brokers. it may create a corresponding number of brokers. These details are stored in the database and fetch from the database and displayed.

Fig4:DataBroker Creation

Virtual Machine Creation

Create a virtual Machine id, Broker id, Ram and bandwidth using Cloud sim tool. In this module user enter the number of virtual Machines .it may create a corresponding number of virtual Machines. These details are stored in the database and fetch from the database and displayed.

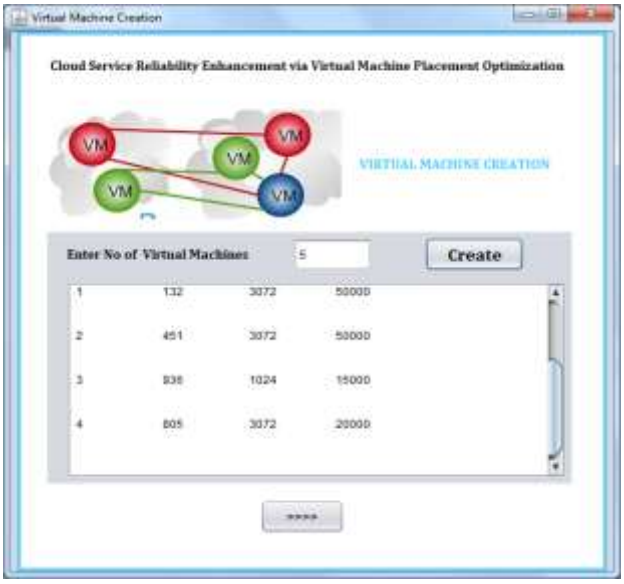

Fig5:Virtual machine creation

Cloudlet Uploading

To upload the Files into the cloud and in that text area to click all the file path then only display the size and stored in the database. Cloudlet file size was also Displayed in the page.

Fig6:Cloudlet Uploading

Virtual Machine Allocation

Files that are allocated to the virtual Machine and contains files size and allocated virtual Machine Id. These details are stored in the database.

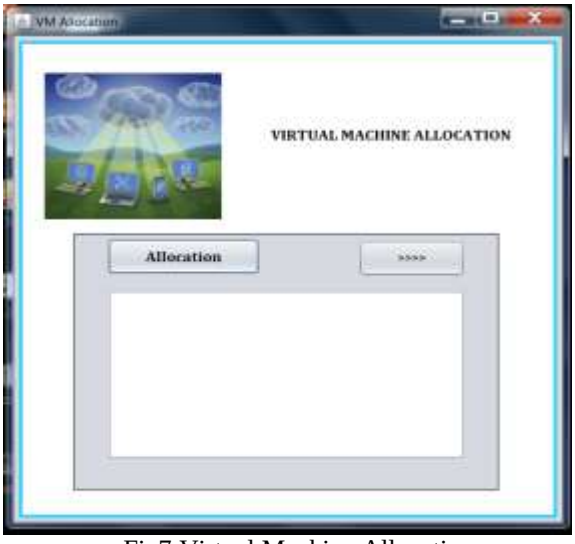

Fig7:Virtual Machine Allocation

Reallocate VM and Backups

If its fault means reallocate the virtual Machine and backups and displayed in the database. Its fetch from the Database and display in the table format.

Fig8:Reallocate Vm

IV. CONCLUSION

This paper aims at enhancing the reliability of server-based cloud services whose fault-tolerance level can be measured and assured in terms of the replication-based k-fault-tolerance metric

We found the fault machine replaced by new Virtual Machines using a novel metric mechanism.

REFRENCES

[1] W. Voorsluys, J. Broberg, and R. Buyya, "Introduction to cloudcomputing,"Cloud computing: Principles and paradigms, pp. 3–37,2011.

[2] W. Zhao, P. Melliar-Smith, and L. Moser, "Fault tolerance middle-ware for cloud computing," inCloud Computing (CLOUD), 2010IEEE 3rd International Conference on, pp. 67–74, July 2010.

[3] I. Goiri, F. Julia, J. Guitart, and J. Torres, "Checkpoint-based fault-tolerant infrastructure for virtualized service providers," in Network Operations and Management Symposium (NOMS), 2010 IEEE,pp. 455–462, April 2010

[4] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, "Survivablevirtual infrastructure mapping in virtualized data centers," inCloud Computing (CLOUD), 2012 IEEE 5th International Conferenceon, pp. 196–203, June 2012.

[5] Z. Zheng, Y. Zhang, and M. Lyu, "Cloudrank: A qos-drivencomponent ranking framework for cloud computing," inReliableDistributed Systems, 2010 29th IEEE Symposium on, pp. 184–193, Oct2010

