An Automated Methodology for Secured User Allocation in Cloud

Srijita Basu(✉), Anirban Sengupta, and Chandan Mazumdar

Jadavpur University, Kolkata, India
srijita.basu202@gmail.com, anirban.sg@gmail.com,
chandan.mazumdar@gmail.com

Abstract. Use of cloud infrastructure by enterprises for hosting their data and applications is growing at a rapid pace. This calls for proper security measures for protecting sensitive enterprise data within the Cloud. Absence of owners' control over data stored in Cloud introduces major security concerns that should be handled by the Cloud Service Provider. One of the most critical aspects is secure handling of data deletion; it should be ensured that sensitive data is completely deleted from the Cloud on an owner's request to avoid data leakages. This paper presents a methodology for effective and efficient allocation of Virtual Machines to users, while ensuring security of sensitive data that belongs to users within the same Conflict of Interest classes. The issue of secured data deletion has been addressed by the proposed methodology which, being a provider-end solution, does not incur any additional user overhead.

Keywords: Cloud security · Conflict of interest · Secured data deletion · Virtual machine reservation

1 Introduction

Cloud can be visualized as a distributed system comprising of a set of virtual machines that can be dynamically provisioned to meet the varying resource requirements of a consumer [1]. The organization/entity that maintains a public cloud is referred to as the Cloud Service Provider (CSP). Cloud relieves an enterprise of the overhead of physical installation and maintenance of its system, which automatically reduces the overall operational cost and enhances system efficiency. The modalities of CSP-Consumer relationship are governed by the terms and conditions defined within a Service Level Agreement (SLA). An underlying concern lies in the fact that the consumer has to rely completely on the CSP for the maintenance of privacy and security of sensitive data and services. The notion of mutual trust is achieved to some extent by negotiating the SLA, but still a good number of cloud-specific security issues become inevitable that need to be handled by either the CSP or the consumer [2].

Maintenance of data security in Cloud poses serious challenges owing to its distributed nature and multi-tenant architecture [3]. The data life cycle comprises of several phases, namely data generation, data storage, data usage, data distribution and data deletion. CSP should support all these phases with appropriate security mechanisms [4]. Most importantly, the process of data deletion is crucial in Cloud; this should

P. Mueller et al. (Eds.): SSCC 2016, CCIS 625, pp. 137–151, 2016.
DOI: 10.1007/978-981-10-2737-6_12

be handled carefully by the CSP to ensure permanent and complete deletion of data on a consumer's request. Moreover, the data backups (scope, saving intervals, saving times, storage duration, etc.) should be transparent and auditable for the consumers. Lack of secure deletion strategies may lead to leakage of sensitive data to unauthorized entities.

Traditional security models fail to address the requirements that are specific to Cloud systems [5]. Recently developed cloud-centric models are either too intricate, or fall short in properly representing the actual state and operations of Cloud systems [6]. Moreover, these models do not address the issue of secure data deletion, which continues to remain an open problem. This paper attempts to address this research gap by proposing a novel methodology that would enable a CSP to avoid data leakage that may occur due to incomplete removal of consumers' data. The proposed methodology also includes a technique for identifying the Virtual Machines (VMs) that are most suited as well as secured with respect to a consumer's requirements.

Rest of the paper is organized as follows. In Sect. 2, a survey of related work is given. Section 3 describes the System model and design goal. It presents the proposed methodology, including the service units involved and the algorithm for secure data removal. Section 4 illustrates the usefulness of the proposed methodology with the help of a case study. Finally, Sect. 5 concludes the paper.

2 Related Work

Several security models and mechanisms for cloud-based services have been proposed in recent years. Some of the significant contributions are surveyed here.

Wang et al. [7] proposed a Privacy-Preserving Public Auditing scheme meant for assuring data integrity/correctness within Cloud storage. Here, CSP is considered to be an untrusted party which may hide data losses or even free storage by deleting the blocks that are rarely accessed by the consumer. A Third Party Auditor and a public key based homomorphic authenticator have been used to prevent such breaches. However, it lacks dynamic file handling capabilities and involves large number of message transfers.

Yu et al. [8] introduced a fine-grained access control scheme for clouds. Each data file is stored in encrypted form, and is associated with a set of attributes. A logical expression is associated with each user which defines her access structure over the attributes, thus identifying the data files that she is allowed to access. However, complications arise when a user is to be removed from the server, which requires the data owner to re-encrypt all the data files accessible to that user.

Liu, Wang and Wu [9] proposed a time-based proxy re-encryption scheme that allows a user's access right to expire automatically after a pre-determined period of time. Each data is associated with an attribute-based access structure and an access time, and each user is identified by a set of attributes and a set of eligible time periods which denote the period of validity of the user's access right. The main drawback of this scheme is that it does not have provision for fine-grained time accuracy.

Wang, He and Tang [10] introduced a Cloud data integrity checking scheme based on identity-based proxy-oriented technique, which eliminates the tedious job of certificate maintenance required for verification. The verification is done based on the tag-block configuration of the stored files, a pseudo-random function and a pseudo-random permutation generated by the system, and the bilinear pairings.

Besides, some research has been carried out on cloud sensing/monitoring schemes which help in automating the provisioning of cloud services [11].

It is obvious that most of the existing cloud security models are either costly or too complex. Moreover, they are mostly consumer dependent, which is quite infeasible in a practical Cloud computing scenario. It is important to develop a comprehensive solution that could address cloud specific data security problems at the CSP end. This paper presents such a consumer-independent data security scheme.

3 Proposed Methodology

The Cloud model that has been presented here is composed of the CSP and the Cloud user or service owner (consumer) who deploys her service or stores her data in the Cloud system. The main aim is to formulate a detailed procedure that should be followed when a user requests for some VM instance to deploy her service. The goal here is to achieve user data confidentiality as well as proper utilization and balance between the VMs such that the most appropriate VM (based on user requirement, security issues as well as present condition of the Cloud system) is allocated to the user efficiently.

Basu et al. [12] addressed security based on the present content of each VM by applying suitable separation policies. This paper enhances that work by considering traces of deleted data before granting access to a particular VM. Thus, one of the most vital areas of Cloud data security, *data deletion and disposal issues*, have been addressed here. The problem with data deletion, when using a Cloud-based service, lies in the fact that when a request for data or service removal is made by a consumer, the CSP usually deletes only the pointers that point to the particular data that need to be deleted. However, problems might ensue when a user is assigned a VM which previously hosted data of another user with whom she has conflict of interest. A malicious user may try to obtain sensitive data of some other enterprise that may reside there, thus posing a threat to data confidentiality. Figure 1 illustrates this scenario with a sample case.

The present work is a small step towards handling such data deletion and access issues efficiently. An important assumption of the proposed methodology is the existence of mutual trust between the CSP and the Cloud user. The CSP is believed to be honest and the security issues that have been considered here are from the perspective of malicious users. The entire procedure from request initiation by a user to fulfillment of the request by allocating necessary resources (i.e. VM) by the CSP involves various functions and data structures. These are described in the following sub-section.

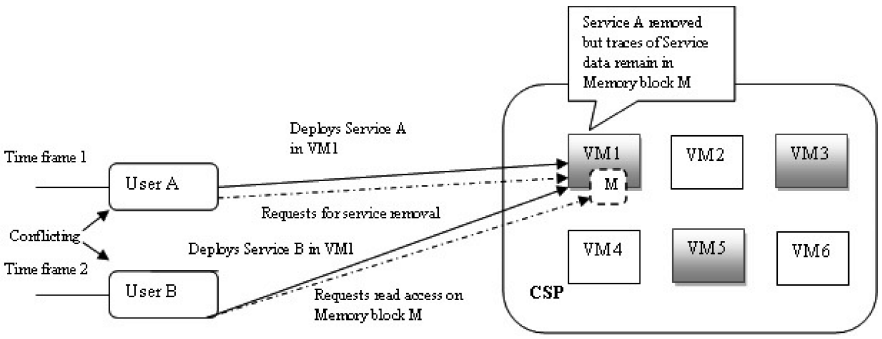

Fig. 1. Cloud with 6 VMs (Grey VMs contain applications/data; white VMs are empty), and 2 users with conflict of interest. Security breach occurs when *UserB* tries to access a particular memory location in VM1.

3.1 Service Units and Data Structures

The proposed methodology comprises of four units, namely **VM Allocation unit**, **Service Automation unit**, **Deletion Analyzer unit** and **Conflict Analysis unit**; they are used for managing VM allocation, VM reservation, complete removal of deleted data and analyses of Conflict of Interest classes [12], respectively.

3.1.1 VM Allocation Unit

This particular module inputs the user requirements (in terms of memory, hardware, storage, etc.) and searches for the best possible set of VMs that are able to meet those requirements. Once such a set is found, an LRU (Least Recently Used) algorithm is used to select a single VM out of this set. Use of LRU algorithm ensures proper and uniform utilization of CSP's resources and helps avoid cloud sprawl or VM sprawl [13].

3.1.2 Conflict Analysis Unit

The VM selected by the **VM Allocation unit** is input to this module which checks for existing conflicts based on Conflict of Interest classes [12]. This module uses the Chinese wall security policy for conflict analyses as suggested in [12]. After a VM passes this unit, it enters the next module.

3.1.3 Deletion Analyzer Unit

This module uses a binary variable *flag*. When deleting enterprise data from a VM, the value of *flag* is set to "true". A list *Org-index* is used to store the names of enterprises whose data have been deleted, along with the pointers which were actually removed from the memory. Both *flag* and *Org-index* are maintained on a per-VM basis. The field "status" in User_details table is set to "absent" and the timestamp value (explained in the next section) of the corresponding entry of the VM in User_reservation queue (if present) is set to zero. If the value of *flag* is "false", the VM is assigned to the user. However, if it is "true", the module checks *Org-index* list to identify names of

conflicting enterprises [12]. If no conflicts exist, the VM can be assigned to the user. On the other hand, if conflicts are detected, the deleted pointers, corresponding to the conflicting enterprises, are identified and the corresponding memory blocks are over-written by some garbage value. After the completion of this process the VM can be safely allocated to the user. This helps to prevent illegal leakages of sensitive data.

3.1.4 Service Automation Unit

In addition to managing secure deletion of data, the proposed methodology also employs efficient means of handling user requests by using a reservation technique. The Service Automation (SA) unit reserves Virtual Machine Images (VMIs) for each user and manages the respective copies of reserved VMIs [14], as follows:

1. When a user request is made for the first time, a suitable VM is assigned to her using the procedure mentioned in the earlier modules. In addition to this, the SA unit constructs a reservation queue for that user containing the presently assigned VMI ID, along with those VMI IDs that have been found to be equally suitable for allocation by the earlier modules.
2. The same VMI may be reserved by the SA unit for more than one user. In such a scenario, any update (i.e. actual allocation of the VM to a particular user) must be reflected in all copies of reservation queues that contain the affected VMI IDs.
3. Periodic checks are conducted by the SA unit in order to detect security conflicts that might arise owing to new allocations. If security conflicts are detected, SA unit associates a status flag with the corresponding VMI ID in the reservation queue and sets its value to “FALSE”. Later, if some VM is de-allocated, as a result of which previous conflicts are removed, the same change is reflected in the user reservation queue during periodic updates by setting the status flag to “TRUE”.
4. When a user request is encountered for the second time, no further configuration checking or security checking is required, as a suitable VM could be assigned to the user readily from the reservation queue. Thus, though the implementation of the SA unit incurs some provider-overhead, it enables the CSP to serve its users efficiently.
5. It has been assumed here that a user follows the same trend of requests, which may not be the case always. It may so happen that the same user now has different requirements. In such a scenario, the user has the freedom to choose a new VM of suitable configuration from the list of available ones.
6. A problem may arise owing to the periodic update of reservation queues that is executed at specified time intervals. It may be the case that a certain allocation is done before the required update has taken place in the local VMI copy, thus resulting in conflicting enterprises sharing the same VM. Such situations can be handled by using a time-stamp. Whenever a VM is allocated to some user, the change is immediately reflected in the main VMI and a time-stamp is associated with the particular VMI (main copy) showing the time of allocation. Later on, when periodic checks are carried out, the change is propagated to all the local copies (residing in the user queues) corresponding to the particular VMI and a new time-stamp is assigned to the main as well as the local VMI copies. Now, when a user requests a VM, the SA unit can automatically allocate one, after checking the time-stamps associated with the local VMI copy in the reservation queue and that of

the main VMI copy. If both the time-stamps are same, the VM is allocated to the user. However, if the time-stamp of the main copy is found to be greater than that of the local copy, it implies that the local copy of the VMI is not updated. In such a scenario, SA unit first synchronizes all local copies of the VMI with the main copy and then performs the necessary allocation.

Figure 2 shows the detailed workflow needed to implement the proposed data deletion and VM allocation scheme, while the algorithms for implementing the methodology are described in the following sub-section.

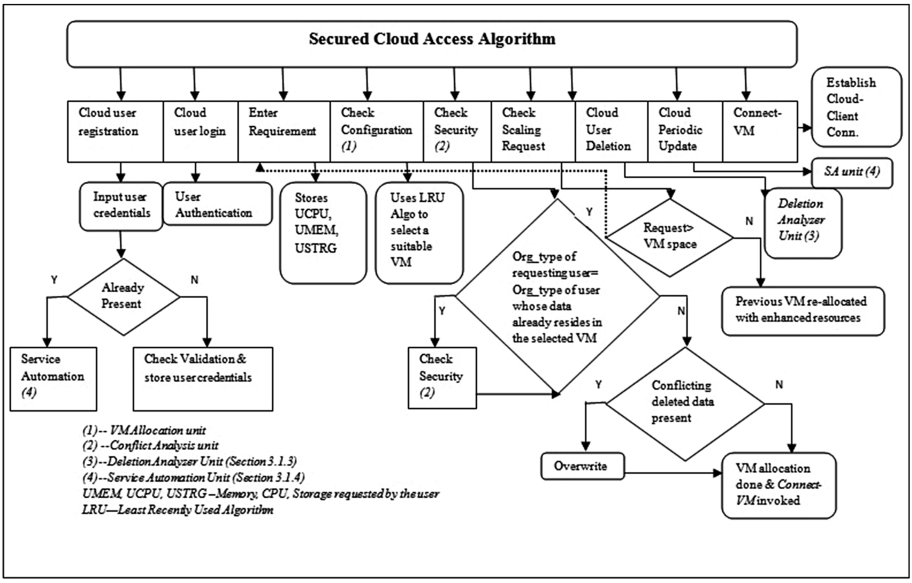

Fig. 2. Flowchart for secured cloud access.

3.2 Algorithm for Secured Cloud Access

Tables 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12 contain various algorithms that can be used to implement the proposed data deletion and VM allocation scheme.

The *Secured Cloud Access* algorithm (Table 1) has been designed to implement the service units described in the previous sub-section. It comprises of several subroutine calls which are detailed in Tables 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12. The entry point of the algorithm is at *Cloud User Registration* and the exit point occurs at line no. 51 or 54 when *Connect-VM(VM_ID)* either establishes a connection between the CSP and the Cloud user, or prints an Error message, as the case may be.

If the total number of VMs in the system is considered to be 'n', then a simple analysis shows that the Worst case Time Complexity of the proposed Algorithm is O(n).

Table 1. Secured cloud access algorithm

1. Name, Mail-id, Org_name, Org_type, password, Re-type password are fields containing the values of the Name, Email address, Organization Name, Organization Type, password (authenticating element), password repetition, respectively of the user who wants to use the Cloud service. Status is an additional field holding values "present" or "absent" signifying that the user is a present Cloud user or a deleted user.
2. UMEM,UCPU,USTRG are fields storing the user memory, CPU, and storage requirements that she wants to avail of the CSP
3. VCPU, VMEM, VSTRG are entities storing the memory, CPU, storage associated with each Virtual machine of the CSP.
4. SCPU, SMEM, SSTRG are the entities storing the memory, CPU and storage that the user wants to scale up.
5. VM_ID is the identifier/variable used to represent each virtual machine of the CSP.
6. VM_list[] is an array storing a list of VMs and integer variables i, j are initialized to 0 which acts as the counter for the lists used.
7. Name′[] and Name″[] are string arrays for storing names of Cloud users.
8. User_details table is a component of the CSP database storing the values of Name, Email address, Organization Name, Organization Type, Status and password of a Cloud user. Primary key of the table is (Name, mail_id).
9. User_requirement table stores the values of Name, Email address, User memory, CPU, and storage requirement. Primary key of the table is (Name, mail_id).
10. VM_allocation table stores the values of Name, Email address, Time_stamp and VM identifier i.e. the VM_ID of the Virtual Machine that has been allocated to the particular user. Primary key of the table is (Name, mail_id) (It has been assumed that a user cannot be allocated more than one VM simultaneously).
11. VM_details table stores the values of VM_ID, Virtual Machine memory, CPU, storage. Primary key of the table is (VM_ID).
12. A variable flag_VMIDn stores either "true" or "false" for each VM.
13. Org-index is a structure storing the list of deleted user names and the list of particular pointers that have been deleted from the VM memory.
14. User_reservation queue stores list of reserved VMIs (as VM_ID), associated Time stamps (TS) and a status flag for each VM_ID with an initial default value "True" for each user. "User_reservation queue.name" represents the name or identifier of each queue i.e. the user name to which the queue belongs.
15. If ***Cloud User Registration*** () = true
16. Var1 = ***Cloud User Login*** ();
17. Else
18. Terminate1=1;
19. End If
20. If Var1 = true
21. If(VM not allocated already)
22. Var3 = ***Enter Requirement***(Name, Mail-id);
23. Else if(VM already allocated)
24. Terminate2=1;
25. Else if(User wants On-demand scaling of resources)
26. Var4 = ***Check Scaling***(Name, Mail-id);
27. End If
28. Else
29. Go to Step 15;
30. End If
31. If Var3 = true
32. If ***Check Configuration***(Name, Mail-id) = true
33. Var5= ***Check Security***(Name, VM_ID, VM_ID);
34. End If
35. End If
36. If Var4 = 0
37. Var3 = ***Enter Requirement***(Name, Mail-id);
38. Go to step 31;
39. Else if Var4 = 1
40. Terminate3=1;
41. End If
42. If Var5 = true
43. Terminate4=1;
44. End If
45. Call ***Cloud User Deletion***(Name, Mail-id);
46. Call ***Cloud Periodic Update***(User_reservation queue U);
47. If (Terminate1=1 or Terminate2=1 or Terminate3=1 or Terminate4=1)
48. Var2 = ***Connect-VM***(VM_ID);
49. End if
50. If Var2 = true
51. End
52. Else
53. Print "Error in Connection"
54. End
55. End if

Table 2. Cloud user registration subroutine

Subroutine: ***Cloud User Registration*** ()
1. *Input* Name, Mail-id, Organization Name, Organization Type, password, re-type password;
2. If ***Check Validation***() = true and Name != "Name" and Mail-id != "Mail-id" in User_details table
3. *Store* Name, Mail-id, Organization Name, Organization Type, "present", password in User_details table;
4. Return True;
5. Else if (Status = "absent" && Name="Name") in User_details_table
6. Call ***Service Automation***(Name, Mail-id)
7. Return False;
8. End If

Table 3. Cloud user login subroutine

Subroutine: ***Cloud User Login*** **()**
1. *Enter* Mail-id and password;
2. *Retrieve* stored values of Mail-id and password from User_details and *store* them as Mail-id′ and password′ ;
3. If (Mail-id′= Mail-id and password = password′)
4. Return true
5. End if

Table 4. Enter requirement subroutine

Subroutine: ***Enter Requirement*** **(Name, Mail-id*)***
1. *Input* UCPU, UMEM, USTRG;
2. *Store* Mail-id, Name, UCPU, UMEM, USTRG in User_requirement table;
3. Return true;

Table 5. Check configuration subroutine

Subroutine: ***Check Configuration*** **(Name, Mail-id*)***
1. If (Name= "Name" and Mail-id= "Mail-id") in User_requirement table
2. *Retrieve* UCPU, UMEM, USTRG from User_requirement table
3. End If
4. If (VCPU >= UCPU and VMEM >= UMEM and VSTRG >= USTRG) in VM_details table
5. *Retrieve* VM_IDs from VM_details table and *store* in VMlist[]
6. End If
7. *Read* VM_ID1;
8. VM_ID1= *LRU* (VMlist[]);
9. *Delete* VM_ID1 from VMlist[]
10. Return true

Table 6. Check security subroutine

```
Subroutine: Check Security (Name, VM_ID1, VMlist[])
1. Read String [] Name′, Name″;
2. If (Name= “Name” and Mail-id = “Mail-id”) in User_details table
3.      Retrieve Organization Type from User_details table and store in Org_type;
4. End If
5. If (Organization Type= “Org_type”) in User_details table
6.      Retrieve Name from User_details table and store in Name′;
7. End If
8. If(VM_ID= VM_ID1)
9.      Retrieve Name from VM_Allocation table and store in Name″
10. End If
11. If (Name′ = Name″)
12.      Message display “ALLOCATION FAILURE”;
13.      Secured= False;
14.      VM_ID2= random (VMlist[]);
15.      Check Security(Name′,VM_ID2, VMlist[]);
16. Else if (flag_VM_ID1= true)
17.      If (Name = “Org_index.name”) in User_details table
18.           Retrieve Organization Type from User_details table and store in Org_type ′;
19.      End If
20.      If (Organization Type= “Org_type” && Name= “Name”)
21.           Retrieve Org_index.pointer
22.           Overwrite memory locations pointed by Org_index.pointer
23.           Secured = True;
24.           Store Name, Mail-id, VM_ID2, TS in VM_allocation table
25.      End If
26. Else if (Secured = True)
27.      Store Name, Mail-id, VM_ID2, TS in VM_allocation table
28. End If
29. while (i < VMlist[].Length and Secured = True)
30.      If(VM_ID = VMlist[i])
31.           Retrieve Name from VM_allocation table and store in Name″
32.      End If
33.      If (Name′!= Name″)
34.          Store VMlist[i] in User_reservation queue of this user
35.          Store User_reservation.TS = current system time
36.      End if
37.      Increment i as i+1
38. End while
39. Return true;
```

Table 7. Connect-VM subroutine

```
Subroutine: Connect-VM (Name, VM_ID)
1. If (Name = “Name”) in User_requirement table
2.      Retrieve UCPU, UMEM, USTRG from User_requirement table ;
3. End If
4. If(VM_ID= “VM_ID”) in VM_details table
5.      Update VM_details table as VCPU= VCPU-UCPU, VMEM=VMEM-UMEM,VSTRG-  USTRG ;
6. End If
7. Establish connection between the Client and the Cloud server and  Return  True;
```

Table 8. Check scaling request subroutine

```
Subroutine: Check Scaling Request (Name, Mail-id)
1. Input SMEM, SCPU, SSTRG;
2. Read VM_ID′;
3. If (Name = “Name”) in VM_allocation table
4.      Retrieve VM_ID from VM_allocation table and store in VM_ID′;
5. End If
6. If (VM_ID = “VM_ID′”) in VM_details table
7.      Retrieve VCPU, VMEM, VSTRG from VM_details table;
8. End If
9. If (VCPU >= SCPU and VMEM >= SMEM and VSTRG >= SSTRG)
10.     Update User_requirement table as UMEM=UMEM+SMEM and UCPU=UCPU+SCPU and USTRG=
   USTRG+SSTRG;
11.     Return 1;
12. Else
13.     If (Name= “Name”)
14.         Retrieve UCPU, UMEM, USTRG from User_requirement table;
15.     End If
16.     If (VM_ID== “VM_ID′”)
17.         Call Cloud User Deletion (Name, Mail-id)
18.     End if
19.     Return 0;
20. End If
```

Table 9. Cloud user deletion subroutine

```
Subroutine: Cloud User Deletion (Name, Mail-id)
1. If (Name = “Name”) in User_requirement table
2.    Retrieve UCPU, UMEM, USTRG from User_requirement table ;
3. End If
4. If (Name = “Name”)
5.    Retrieve VM_ID from VM_allocation table
6. End If
7. If (VM_ID== “VM_ID′”)
8.    Update VM_details table as VCPU = VCPU+UCPU and VMEM=VMEM+UMEM and VSTRG+  USTRG
9. End if
10. If (Name== “Name”)
11.    Delete entry from VM_allocation
12. End if
13. flag_VM_ID= true
14. Org_index.name= Name
15. Org_index.pointer= pointers to be deleted /*pointer addresses holding the data of the deleted user*/
16. Delete the necessary pointers
17. If (Name== “Name”)
18.    Update User_details table as Status=”absent”
19. End if
20. If(VM_ID=User_reservation.VM_ID)
21.    User_reservation.TS=0
22. End if
```

Table 10. Service automation subroutine

```
Subroutine: Service Automation (Name, Mail-id)
1. Search User_reservation.VM_ID of user "Name"
2. Retrieve VM_ID from VM_Allocation table
3. If("User_reservation.VM_ID"!=VM_IDand"User_reservation.TS"!=0andUser_reservation.status==True)
4.      Store Name, Mail-id, VM_ID, TS in VM_allocation table
5. Else if ("User_reservation.VM_ID"=VM_ID and (User_reservation.status = False or "User_reservation.TS"=
   0) and end of queue not reached)
6.      Move to next entry in User_reservation queue
7.      Go to Step 3
8. End if
9. If (end of queue reached and "User_reservation.VM_ID" = VM_ID)  in VM_allocation table
10.     Retrieve TS from VM_allocation table
11.     If (User_reservation.TS < TS)
12.        User_reservation.TS = TS
13.        If  "success" returned by  Cloud Periodic Update ()
14.           Store Name, Mail-id, VM_ID, TS in VM_allocation table
15.        End if
16.     Else if (User_reservation.TS>= TS)
17.        Store Name, Mail-id, VM_ID, TS in VM_allocation table
18.     End if
19. End if
20. If(end of queue reached and  "User_reservation.TS"=0 )
21.     If(Name = "Org_index.name") in User_details table
22.        Retrieve Organization Type from User_details table and store in Org_type ;
23.     End if
24.     If (Organization Type= "Org_type" && Name= "Name")
25.        Retrieve Org_index.pointer
26.     End if
27.     Overwrite memory locations pointed by Org_index.pointer
28.     Store Name, Mail-id, VM_ID2, TS in VM_allocation table
29. End If
```

Table 11. Cloud periodic update subroutine

```
Subroutine: Cloud Periodic Update ()
1. Retrieve TS, VM_ID from VM_Allocation table and store in TS_list[] and VMlist[]
2. While i < TS_list[].length and j< VMlist[].length
3.      If (User_reservation.VM_ID =VMlist[j])
4.          Update User_reservation as User_reservation.TS= max(TS_list[i])
5.          Retrieve Name from VM_Allocation table and store in Name′
6.          If (Name= "Name′")
7.              Retrieve Organization Type from User_details table and store in Org_type;
8.          End if
9.          If (Name = User_reservation queue .name)
10.             Retrieve Organization Type from User_details table and store in Org_type′;
11.         End if
12.         If (Org_type = Org_type′)
13.             Set User_reservation.status=False;
14.             Return "failure"
15.         Else if (Org_type!=Org_type′)
16.             Set User_reservation.status=True;
17.             Return "success"
18.         End if
19.     End if
20.     Increment i as i+1
21.     Increment j as j+1
22. End while
```

Table 12. Check validation subroutine

Subroutine: ***Check Validation ()***
1. If (Name!=NULL and Mail-id!=NULL and Organization Name!=NULL and Organization Type !=NULL and password!=NULL and password = Re-type password)
2. Return true;
3. End If

4 Case Study

The proposed scheme is illustrated with the help of the following case study. The present state of the Cloud server is depicted in Tables 13, 14, 15, 16 and 17.

Table 13. User_details table

Name	Email address	Organization name	Organization type	Status
UserA	usra@abc.com	XYZ services	CRM (Customer Relationship Management)	Present
UserB	usrb@abc.com	PQR services	ERP (Enterprise Resource Planning)	Absent

Table 14. User_requirement table

Name	Email address	User memory	User CPU	User storage
UserA	usra@abc.com	1 GB	2vCPU	20 GB
UserB	usrb@abc.com	2 GB	1vCPU	15 GB

Table 15. VM_allocation table

Name	Email address	VM_ID	Time stamp
UserA	usra@abc.com	14	2016/04/14 14:22:13.656008

Table 16. VM_details Table (including *Flag* and *Org-index*)

VM_ID	VM memory	VM CPU	VM storage	Flag	Org-index
14	1 GB	1 vCPU	0 GB	False	–
23	4 GB	2 vCPU	30 GB	True	UserB [0xFFFF:000F, 0 × 9FFF:000F]
17	4 GB	2 vCPU	20 GB	False	–
9	4 GB	2 vCPU	15 GB	False	–
21	4 GB	2 vCPU	25 GB	False	–

Table 17. State of User_reservation queue

Name	Queue entry 1 (VM_ID, Time Stamp, status)	Queue entry 2 (VM_ID, Time Stamp, status)	Queue entry 3 (VM_ID, Time Stamp, status)
UserA	14, 2016/04/14 14:22:13.6,T	17, 2016/04/14 14:22:13.6,T	9, 2016/04/14 14:22:13.6,T
UserB	23,0,T	21, 2016/04/14 17:25:18.6,T	17, 2016/04/14 17:25:18.6,T

A new user, *UserC* wants to deploy her ERP application in Cloud. Tables 18 and 19 show the user details and requirements, respectively, after the subroutines *Cloud User Registration* and *Enter Requirement* (stated in Sect. 3.2) have been invoked.

Table 18. User_details table

Name	Email address	Organization name	Organization type	Status
UserA	usra@abc.com	XYZ services	CRM (Customer Relationship Management)	Present
UserB	usrb@abc.com	PQR services	ERP (Enterprise Resource Planning)	Absent
UserC	usrc@abc.com	PQR services	ERP (Enterprise Resource Planning)	Present

Table 19. User_requirement table

Name	Email address	User memory	User CPU	User storage
UserA	usra@abc.com	1 GB	2vCPU	20 GB
UserB	usrb@abc.com	2 GB	1vCPU	15 GB
User C	usrc@abc.com	2 GB	1 vCPU	28 GB

The algorithm checks for a suitable VM by invoking the subroutine *Check Configuration.* It is evident that VM-23 is the only one capable of meeting the user requirements (Table 16). VM-23 is now checked for security compliance by the subroutine *Check Security.* No conflicting enterprise data is found but the value of *flag* is "true". As is evident from Table 16, *Org-index* contains *UserB* as an entry whose enterprise type is "ERP" which is the same as that of *UserC*. Therefore, the memory locations stored in *Org-index* are first overwritten and then VM-23 is allocated to *UserC* with the corresponding entry being done in VM_allocation table. It should be noted here that the User_reservation queue for *UserC* contains only VM-23, since this is the only one which matches the configuration requirements of the user. The post-allocation changes in the Cloud system are depicted in Tables 20, 21 and 22.

Thus, the case study illustrates the functions of the proposed scheme.

Table 20. VM_allocation table

Name	Email address	VM_ID	Time stamp
UserA	usra@abc.com	14	2016/04/14 14:22:13.656008
UserC	usrc@abc.com	23	2016/4/15 12:46:18:656001

Table 21. VM_details table (including *Flag* and *Org-index*)

VM_ID	VM memory	VM CPU	VM storage	Flag	Org-index
14	1 GB	1 vCPU	0 GB	False	–
23	2 GB	1vCPU	2 GB	True	UserB [0xFFFF:000F, 0 × 9FFF:000F]
17	4 GB	2 vCPU	20 GB	False	–
9	4 GB	2 vCPU	15 GB	False	–
21	4 GB	2 vCPU	25 GB	False	–

Table 22. State of User_reservation queue

Name	Queue entry 1 (VM_ID, Time Stamp,status)	Queue entry 2 (VM_ID, Time Stamp,status)	Queue entry 3 (VM_ID, Time Stamp,status)
UserA	14, 2016/04/14 14:22:13.6, T	17, 2016/04/14 14:22:13.6, T	9, 2016/04/14 14:22:13.6, T
UserB	23,0,T	21, 2016/04/14 17:25:18.6,T	17, 2016/04/14 17:25:18.6,T
UserC	23, 2016/4/15 12:46:18:6,T	–	–

5 Conclusion and Future Work

In this paper, a novel methodology for Cloud system security has been proposed. The different service units and their functions have been described. An algorithm for implementing the functionalities of the proposed scheme has been detailed. Cloud operations have been described considering aspects of security, data deletion and access control. The methodology can be used to manage the operations and security aspects of cloud services smoothly, and provide assurance to users about the safety of hosting such services. It ensures that a user is not able to access data that belongs to an enterprise within the same Conflict of Interest class [12] as that of the requesting user. This helps to protect the confidentiality of enterprise data. A detailed case study has been included to demonstrate the utility of the work.

The main overhead of this scheme lies in implementing the subroutine, *Cloud Periodic Update* which is to be executed at regular intervals of time for assuring updated and safe copies of VMI. An erroneous update would result in an incorrect and

unbalanced state of the Cloud system leading to improper allocation and de-allocation of the VMs and, possibly, security issues. Moreover, the assurance made by the scheme of overwriting memory locations of conflicting datasets would need hardware-level intervention which should be done carefully, avoiding other unrelated memory locations from getting affected. Future work is geared towards the development of an automated tool based on the proposed methodology. This would help in eliminating such errors while implementing the proposed operations.

References

1. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. J. Future Gener. Comput. Syst. **25**, 599–616 (2008)
2. Hashizume, K., Rosado, D.G., Fernández-Medina, E., Fernandez, E.B.: An analysis of security issues for cloud computing. J. Int. Serv. Appl. **4**(5), 1–13 (2013)
3. Rong, C., Nguyen, S.T., Jaatun, M.J.: Beyond lightning: a survey on security challenges in cloud computing. J. Recent Adv. Technol. Theor. Grid Cloud Comput. Bio-Eng. **39**(1), 4–54 (2013)
4. Sun, Y., Zhang, J., Zhiong, Y., Zhu, G.: Data security and privacy in cloud computing. J. Dist. Sens. Netw. **2014**, 1–9 (2014)
5. Subashini, S., Kavitha, V.: A survey on security issues in service delivery models of cloud computing. J. Netw. Comput. Appl. **34**(1), 1–11 (2010)
6. Majumder, A., Namasudra, S., Nath, S.: Taxonomy and classification of access control models for cloud environments. In: Zaigham, M. (ed.) Continued Rise of the Cloud: Advances and Trends in Cloud Computing. Computer Communications and Networks, Springer. London (2014)
7. Wang, C., Wang, Q., Ren, K., Louu, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: 2010 Proceedings of INFOCOM, pp. 1–9. IEEE Press, San Diego (2010)
8. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: 2010 Proceedings of INFOCOM, pp. 1–9. IEEE Press, San Diego (2010)
9. Liu, Q., Wang, G., Wu, J.: Time-based proxy re-encryption scheme for secure data sharing in a cloud environment. J. Inf. Sci. **258**, 35–370 (2014)
10. Wang, H., He, D., Tang, S.: Identity-based proxy-oriented data uploading and remote data integrity checking in public cloud. J. IEEE Trans. Inf. For. Secur. **11**(6), 1165–1176 (2016)
11. Aceto, G., Botta, A., de Donato, W., Pescapè, A.: Cloud monitoring: definitions, issues and future directions. In: 2012 Proceedings of IEEE CloudNet, pp. 63—67. IEEE Press, Paris (2012)
12. Basu, S., Sengupta, A., Mazumdar C.: Implementing Chinese wall security model for cloud-based services. In: ICGCIoT 2015, pp. 1083–1089. IEEE Press, Noida (2015)
13. Cloud Management and Monitoring. http://searchcloudcomputing.techtarget.com/definition/cloud-sprawl
14. Infrastructure as a Service Cloud Concepts. http://www.ibmpressbooks.com/articles/article.asp?p=1927741&seqNum=7