International Journal of Modern Trends in Engineering and Research
www.ijmter.com
e-ISSN No.:2349-9745, Date: 2-4 July, 2015

Provisioning Rapid Elasticity by Light-Weight Live Resource Migration

S. Kirthica[1],Rajeswari Sridhar[2]
[1]*Department of Computer Science and Engineering, Anna University, Chennai, s.kirthica@gmail.com*
[2]*Department of Computer Science and Engineering, Anna University, Chennai, rajisridhar@gmail.com*

Abstract-This paper proposes an algorithm to perform live migration of resources of a VM (storage and computing power) as demands rise or fall with minimal service interruption and negligible performance impact. Resources, distributed across several Nodes, are provided to users on demand. When a request for additional resource is received, the user's current node is checked to see if it can accommodate this additional request. If not, then the user's resources are migrated to another node which can satisfy the user's request. Resources are migrated by synchronizing disk contents of the user in another node, rolling back the executing processes during migration in the current node, reconnecting user to the other node and restarting process execution. Evaluation using parameters like response time, service unavailability and number of failed requests show that uninterrupted service is provided.

Keywords-cloud computing; elasticity; live migration; resource migration; rapid elasticity; light weight live migration; VM migration; live VM migration; virtual machine migration

I. INTRODUCTION

Cloud computing is a technology that provides hardware, networks, storage, services and interfaces as a service [1]. This service includes providing software, infrastructure, and storage, either as separate components or a complete platform to meet the user's demand. The key characteristics of cloud computing include on-demand self-service, ubiquitous network access, location and device independence, rapid elasticity and pay-per-use [1].

The most attractive feature of the cloud is resource scaling in and scaling out as per user's demand which is referred to as elasticity [2]. With elasticity, organizations need not concern themselves about wasting resources on services whose popularity does not meet their predictions by over-provisioning or losing potential customers and hence the revenue due to under-provisioning [3].

A solution for elasticity of resources is automatic live migration which allows a cloud administrator to move a running virtual machine or application between different physical machines without disconnecting the client or application.

This paper is organized as follows: Section 2 briefs the existing works in elasticity and live migration. Section 3 deals with the architecture of elasticity followed and the flow of activities involved in migration of resources of an instance by describing the algorithms involved. The implementation details of the project are discussed in section 4. The time complexity of the algorithms is explained in section 4 along with the evaluation parameters. Section 5 concludes the paper and discusses the future work.

II. RELATED WORK

Adaptive live migration of virtual machines is discussed by Yi Zhao and Wenlong Huang [4]. In this work, the time taken for virtual machine migration is reduced by sharing the storage. The relationship between VLAN and virtual machines is maintained throughout migration. The

requirements are managed in a distributed fashion based on sampling. In the work by Kejiang Ye et al. [5], multiple virtual machines are migrated in a live manner with different resource reservation methods using a framework consisting of 4 players. The first player, Migration Decision-Maker, as the name suggests, has the most important task of deciding about migration. The actual process of migration is controlled by the second player, the Migration Controller. The third player, Resource Reservation Controller, is responsible for implementing the different resource reservation strategies for both the source and target machines of migration. The status of all the resources is monitored by the fourth player, Resource Monitor. This is essential to make decisions on migration. These two works deal with only live migration of virtual machines which is bulky and time consuming and will cause performance impacts [4], [5].

In one of the work, Albatross [6], migration of database cache and the state of active transactions is carried out. This had ensured the researchers minimal impact on transaction execution while allowing the transactions active during migration to continue execution. Though this work provides for live migration of resources, it only migrates storage power i.e. DBMS.

Sadeka Islam et al. [7] discuss improved ways to quantify the elasticity concept by using data available to the consumer. The financial penalty a consumer has to pay in the case of under-provisioning and over-provisioning is seen here. Here the provider is subject to a higher operating cost than necessary.

Based on the disadvantages in the existing works that has been discussed in this section, a new algorithm has been proposed to perform light-weight live migration of resources as live migration of virtual machines is bulky and consumes a lot of time [6]. Hence in this paper, a solution for the problem faced in live migration of virtual machines is provided. The proposed solution reduces the overhead caused by migration of the entire virtual machine in terms of response time and service unavailability.

III. SYSTEM DESIGN

A. Elasticity Architecture

The architecture of cloud elasticity is given in figure 1. It consists of the following components: User, Resource Locator (RL), Node, Instance and Controller.

User refers to the one requesting for resources from a cloud. An *Instance* is the resource allocated in a node for a user's request. This instance will be available in any of the nodes that are in the cloud. All the resources available in the cloud are provided by a cluster of nodes. A *Node* may provide resources for several instances. There is no specific node in which an instance can reside. Whenever an instance is created, an entry is made in the *Resource Locator table* (RL). This table (RL) is used to map user information to the node in which the user's resources are located. Any change in information of the instance is updated in RL. The table is also updated with the details of migration of resources utilized by an instance.

The decision on migration is made by the *Controller* who decides on the following:

- The instance to be migrated
- when the instance has to be migrated
- which node is the source node for migration
- which node is the destination node for migration

When a request for additional resource is made by a user of an instance, RL is queried to get information of the disk in which the instance resides. The node in which the instance resides is checked to see if it has the additional resources requested by the instance. If so, the additional

resources are provided and the instance resides in the same node. If the requested amount of resources is not available in the node in which the instance resides then the instance has to be migrated to a node (destination) which can accommodate the available resources of the instance and will be able to provide the additional resources required. Then the resource from the destination node is allocated to the instance. In this manner, elasticity is achieved. It is to be noted that an instance cannot reside in more than one node.

B. Migration Flow

Once the controller initiates migration, a new instance is created in the destination node. An initial snapshot of the instance in the source node is taken. This snapshot shows the contents of the disk i.e. the entire root directory of the instance. It is copied to the newly created instance in the destination node. The source instance continues to run the processes already running in it. This will lead to a lag in the state of the destination instance when compared to the source instance. This lag in the state of the destination instance compared to the source instance is noted. Then this change in state alone is copied to the destination instance instead of copying the entire state. This reduces the time for synchronizing the resources between the source and the destination instances. This has been termed as incremental update of state. This process has to be iterated continuously till there is no change in state of the source instance. If the state keeps changing indefinitely, then the incremental update of the state of the instance is carried for a certain threshold after which this process is terminated i.e. state transfer is performed till there is no change in state between two successive iterations or till a maximum number of iterations have completed.

Once the state of the instance is synchronized, the source node revokes the resources provided to the instance. The ownership of the instance is transferred to the destination node. This migration of instance has to be updated in the resource locator table. All these processes will take place atomically to ensure integrity. This completes the process of migration of the instance from the source node to the destination node. We have designed the following algorithms to support elasticity.

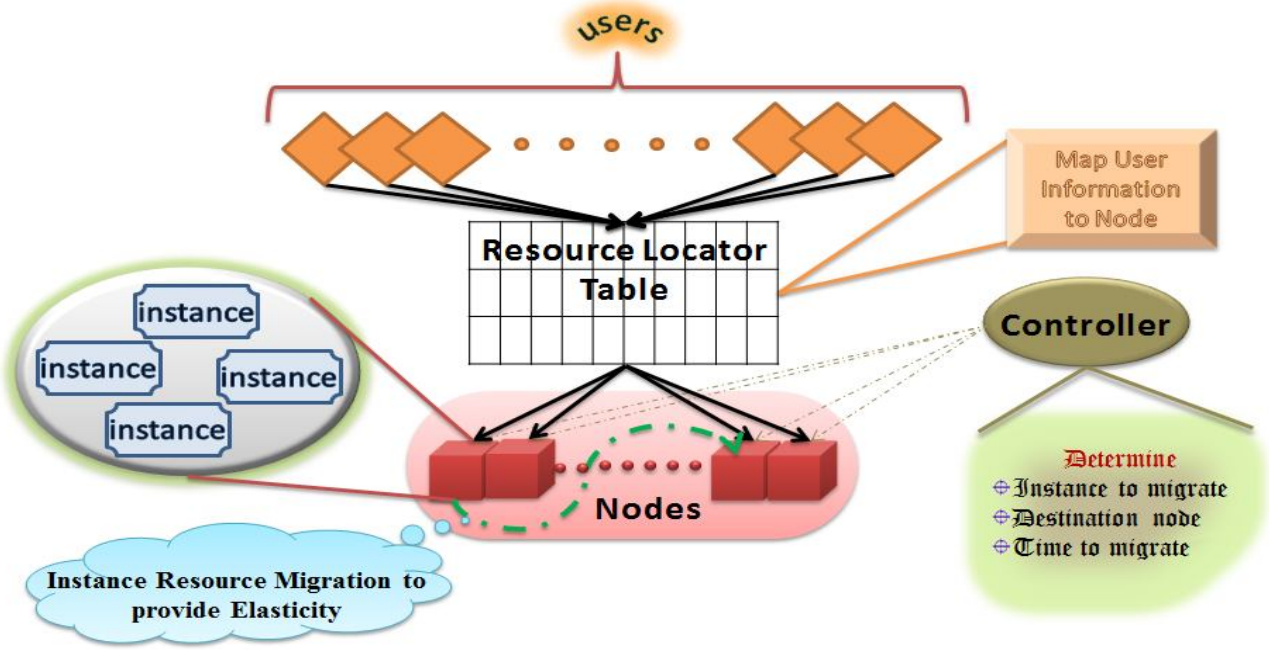


Fig : Elasticity Architecture

Overall Algorithm for increasing the resource of an instance.

```
add_resource_to_EBIn
Input: Instance to be migrated (EBIn), Resource (RSrc)
Output: { EBIn becomes EBIn_E     if Resource is available
        { Resource not added to EBIn          Otherwise
begin
  Compute S = size of EBIn_E = size of EBIn + size of RSrc
  Compute T = Amount of utilized resources in N_EBIn if EBIn = EBIn_E
                                  //N_EBIn is node in which EBIn resides
  if(T >N^max )                   //N^max is maximum amount of Rsrc in N_EBIn
  then
    for each N_i ∈ ( N - N_EBIn)      //N is set of available nodes in cloud
    begin
      Compute T_i = Amount of utilized resources in N_i
      T_i(EBIn) = T_i + S
      if (T_i(EBIn)<= N_i^max )
      then
        N_dst = N_i
        break
      end if
    end for
    if∃N_dst
    then
      callmigrate_instance_resource ( EBIn, N_dst )
    else
      No node can accommodate EBIn_E
      exit ( fail )
    end if
    N_EBInE = N_dst
  else
    N_EBInE = N_EBIn
  end if
  EBIn_E = EBIn + RSrc_NEBInE
  Update in RL
end.
```

Algorithm for migrating resources of an instance from one node to another.

```
migrate_instance_resource
Input: EBIn, N_dst
Output: EBIn terminates and resources of EBIn moved to N_dst as EBIn_new
begin
  Get details of EBIn from RL
  EBIn_new = new instance created in N_dst
  Copy D_EBIn to D_EBInnew
  if ( EBIn is in use )
```



```
    then
      while ( threshold not reached )
      begin
        if (DEBIn == DEBInnew )
        then
          break [while]
        end if
        Synchronize DEBInnew with DEBIn
      end while
      Freeze all processes running in every EBIn session
      Log out of all sessions of EBIn
      Synchronize DEBInnew with DEBIn
      callreconnect_instance ( EBIn, EBInnew )
    end if
    callterminate_instance ( EBIn )
  end.
```

Algorithm for reconnecting an instance.

```
  reconnect_instance
  Input: EBIn, EBInnew
  Output: Processes resumed in EBInnew
  begin
    ∀sessions of EBIn
    begin
      login to EBInnew
      resume frozen status in EBInnew
    end
  end.
```

IV.IMPLEMENTATION AND RESULTS

A. Machine Configuration

The open source software, Eucalyptus [8], for building private cloud computing environments is used for demonstrating this work. Eucalyptus Faststart v3.3.0.1 is installed in Frontend and Node Controllers configuration. i.e. a Eucalyptus cloud with all *Frontend* components on a single system, and three *Node Controllers* (NC) on separate machines.

To set up the cloud, the frontend and the three node controllers are installed in separate partitions in four machines, each having Intel (R) Core™ i7-3770 processor. The partition in each machine has disk capacity of 500 GB, RAM capacity of 3.40 GB, RAM frequency of 3.39 GHz and CPU frequency of 3.40 GHz.

B. Time Complexity

Creation and deletion of an instance takes constant time for completion. The time required for the algorithm *reconnect_instance* depends on the number of sessions (m) logged into the instance EBIn. The time required for the algorithm *migrate_instance_resource* depends on the following factors:

- Threshold or the maximum number of iterations for which synchronization of the disk contents of instances in source and destination nodes should be carried.
- Time required for reconnection.

The time required for algorithm add_resource_to_EBIn depends on the number of nodes available in the cloud (n) and the time required for migration. A consolidated list of time complexity of all the procedures is given in Table 1.

Table 1. Consolidated list of Time Complexities of all the Algorithms

Algorithm Name	Time Complexity
create_instance	Θ (1)
terminate_instance	Θ (1)
add_resource_to_EBIn	O (max {threshold, m})
migrate_instance_resource	O (max {n, threshold, m})
reconnect_instance	O (m)

C. Test Scenarios

The following are a few test scenarios:

1. Migration is not necessary
2. No session of the instance to be migrated is opened and instance is created in first trial.
3. One session of the instance is opened, no user processes are running and instance is created in first trial.
4. One session of the instance is opened, one user process is running and instance is created in first trial.
5. No session of the instance to be migrated is opened and instance is created in second trial.
6. One session of the instance is opened, no user processes are running and instance is created in second trial.
7. One session of the instance is opened, one user process is running and instance is created in second trial.
8. No session of the instance to be migrated is opened and instance is created in third trial.
9. One session of the instance is opened, no user processes are running and instance is created in third trial.
10. One session of the instance is opened, one user process is running and instance is created in third trial.
11. Two sessions of the instance are opened, no user processes are running and instance is created in second trial.

D. Evaluation Parameters

In our work, the following three parameters are used for evaluating elasticity:

- Response time in seconds - Time difference between the initial time a request is raised for additional resource and the time at which the resource is available to the user of an instance.
- Service unavailability in seconds - Change in transaction latency observed by the user of the instance as a result of reconnection.
- Number of failed requests (count) - Number of requests which fail due to unavailability of resource in the cloud.

E. Results and Justification

The relationship between response time, time taken for migration of resources of an instance and service unavailability is shown in the figure 2. It shows that if there is migration of resources on a request for additional resource, the response time is purely based on the time taken for migration. It can also be seen that service unavailability is very low. Hence migration of the resources of an instance to a different node to satisfy a request is not noticeable by the user.

The definition of response time implies that it is the time required for adding the additional resource to the instance. The impact on response time due to change in number of sessions and the number of processes opened is shown in figure 3 (a) and figure 3 (b) respectively.

The definition of service unavailability implies that it is the time required for reconnection of instance. The impact on service unavailability due to change in number of sessions and the number of processes opened is shown in figure 4 (a) and figure 4 (b) respectively. This shows that the service unavailability is very low. Hence, a user of an instance does not notice much lag in a process' execution time when the instance is being reconnected to another instance.

Requests are of two types: new request from a user for resources for the first time and request for additional resources by a user of an instance. The number of failed requests will be high when all the physical resources available in the cloud are currently being utilized. However, in this work, resource allocation of a single user to multiple nodes has not been considered. So, if the existing nodes could not accommodate the rising demand of a user, the request for new resources would fail.

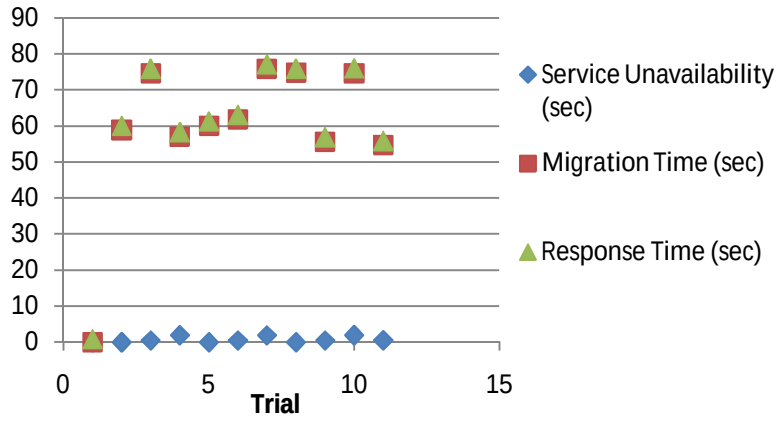
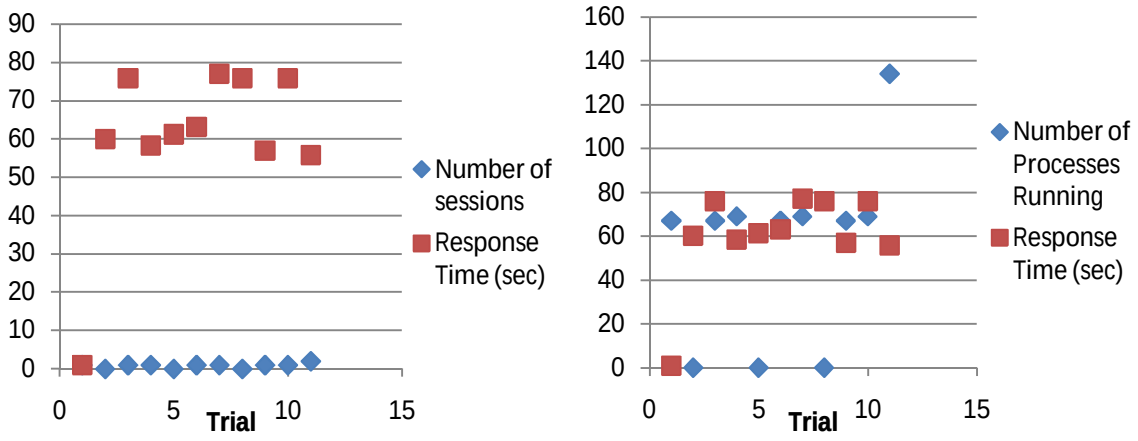
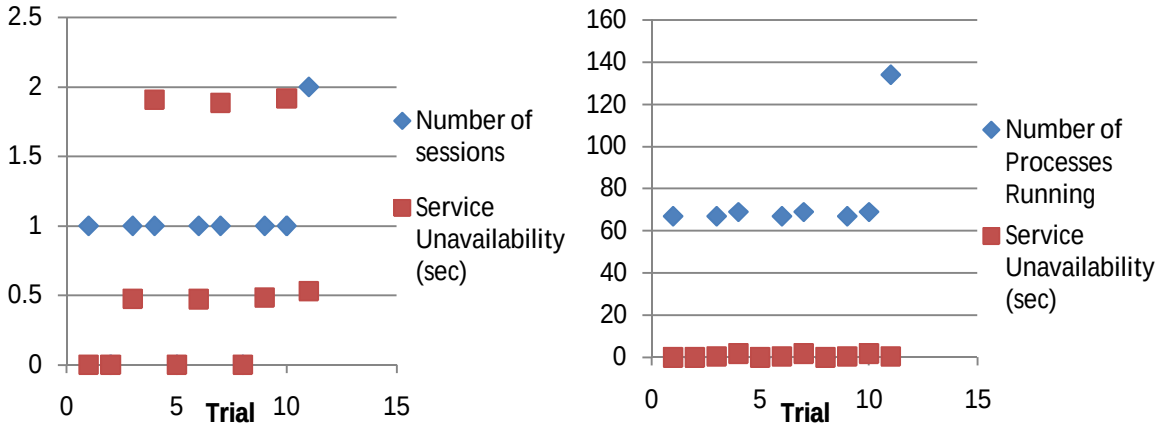


Fig : Evaluation parameters and their interdependence



(a) *(b)*

Fig : Impact on response time due to change in (a) number of sessions and (b) number of processes



(a) *(b)*

Fig 4: Impact on service unavailability due to change in (a) number of sessions and (b) number of processes

CONCLUSION

As the need for space and speed continues to rise, the demand for a faster and spacious environment also grows. This paper presents a solution to provide for these by way of live migration of resources across users. This ensures minimal performance impacts, effective load balancing and efficient resource sharing. The evaluation parameters show the effectiveness of this method.

Capturing the status of RAM poses a security threat to the instance. This threat can be handled while securing the elasticity provided by resource migration. Also, elasticity is provided only for a private cloud environment. Hence a similar solution for public cloud environment can be given.

REFERENCES

[1] G. Shroff,"Enterprise Cloud Computing Technology, Architecture, Applications", 1st edn, Cambridge University Press, 2010.

[2] D. Owens, "Securing Elasticity in the Cloud", Communications of the ACM, Vol. 53.6, pp. 46-51, 2012.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, A. Stoica, and M. Zaharia,"A View of Cloud Computing", Communications of the ACM, Vol. 53.4, pp. 50-58, 2010.

[4] Y. Zhao, and W. Huang, "Adaptive Distributed Load Balancing Algorithm Based on Live Migration of Virtual Machines in Cloud", INC, IMS and IDC. NCM, Fifth International Joint Conference on IEEE,pp. 170-175, 2009.

[5] K. Ye, x. Jiang, D. Huang, J. Chen, and B. Wang,"Live migration of multiple virtual machines with resource reservation in cloud computing environments", Cloud Computing (CLOUD), IEEE International Conference, pp. 267-274, 2011.

[6] S. Das, S. Nishimura, D. Agrawal, and A. E. Abbadi,"Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration", Journal Proceedings of the Very Large Data Base (VLDB) Endowment, Vol. 4.8,pp. 494-505, 2011.

[7] S. Islam, K. Lee, A. Fekete, and A. Liu, "How A Consumer Can Measure Elasticity for Cloud Platforms", Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering, ACM,pp. 85-96, 2012.

[8] D. Johnson,K. Murari, M. Raju, R. B. Suseendran, and Y. Girikumar,"Eucalyptus Beginner's Guide-UEC Edition", CSS Corp, 2010.

