Distributed Resource Allocation in Cloud Computing Using Multi-Agent Systems

Artan Mazrekaj, Dorian Minarolli, and Bernd Freisleben

***Abstract*—The Infrastructure-as-a-Service model of cloud computing allocates resources in the form of virtual machines that can be resized and live migrated at runtime. This paper presents a novel distributed resource allocation approach for VM consolidation relying on multi-agent systems. Our approach uses a utility function based on host CPU utilization to drive live migration actions. Experimental results show reduced service level agreement violations and a better overall performance compared to a centralized approach and a threshold-based distributed approach.**

***Keywords* —Cloud computing, live migration, multi-agent systems, utility function, virtual machines.**

I. Introduction

CLOUD computing is based on sharing computing resources to provide dynamically scalable infrastructures for application execution and data storage. The fundamental service models in cloud computing are: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

Virtualization technology has become commonplace in cloud computing by allowing the creation of multiple virtual machines (VMs) on the underlying hardware to improve resource utilization [1].

Resource allocation in cloud infrastructures is a difficult problem because of system complexity, heterogeneity of resource types, scale of data centers, variability and unpredictability of workloads, and different objectives from different stakeholders in the cloud ecosystem [3].

The rapidly growing demand from hundreds of millions of end users for the use of Internet-scale applications has caused cloud providers (such as Google, Amazon, and Microsoft) to operate large-scale data centers around the world. These large-scale datacenters consume a large amount of energy. A large energy consumption leads to high costs and to high carbon emissions. Currently, data centers that power Internet-scale applications consume about 1.3% of the worldwide electricity supply, and this fraction is expected to grow to 8% by 2020 [17].

Paper received April 24, 2017; revised October 31, 2017; accepted November 5, 2017. Date of publication December 25, 2017. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Aleksandar Nešković.

A. Mazrekaj is with the Faculty of Contemporary Science and Technologies, South East European University, Tetovo, Macedonia (corresponding author e-mail: am23614@seeu.edu.mk).

D. Minarolli is with the Faculty of Information Technology, Polytechnic University of Tirana, Tirana, Albania (e-mail: dminarolli@fti.edu.al).

B. Freisleben is with the Department of Mathematics and Computer Science, University of Marburg, Marburg, Germany (e-mail: freisleb@informatik.uni-marburg.de).

Given the growing interest in addressing these issues, we propose an approach to improving data center efficiency and overall performance, using multi-agent systems.

One of the important approaches that provides major benefits for data centers is live migration of virtual machines, from one physical host to another physical host. Live migration of VMs offers the possibility for allocating resources to running services without interruption during the migration process, which is important for services with particular quality of service (QoS) requirements [2].

Most of existing state-of-art VM resource allocation approaches are centralized,but a centralized controller does not scale well for large cloud infrastructures, might represent a communication bottleneck, and is a single point of failure in terms of reliability[5].

In this paper, we present a novel distributed VM resource allocation approach that uses a utility function based on multi-agent systems. Agents, attached to each physical host, are responsible for making decisions for the live migration of VMs from one host to another host. The key novel feature of the proposed approach is that allocation decisions are based on the individual agents' utility functions, which offers the flexibility of easily changing the allocation policy. Experimental results show that the utility-based distributed resource allocation approach achieves a better overall performance compared to a centralized approach and a threshold-based distributed approach.

The paper is organized as follows. Section II describes related work on dynamic consolidation of virtual machines.Our system architecture is presented in Section III. In Section IV, we discuss centralized and threshold based distributed approaches. Our proposed approach is presented in Section V. Experimental results are presented in Section VI. Section VII concludes the paper and outlines areas for future work.

II. Related Work

In recent years, a main focus of research in the field of cloud infrastructures is to reduce energy consumption and service level agreement (SLA) violations for efficiently managing resources.

Most existing algorithms for VM consolidation [6-8] have as their main objective the energy efficiency and the reduction of SLA violations. For VM consolidation, these approaches use live migration actions. Some authors have treated the VM consolidation process as an optimization problem [15], taking into account constraints such as data center capacity and SLAs.

Nurmi et al. [16]have proposed Eucalyptus, an open source cloud computing frameworkfor VM creation and

resource control in a hierarchical manner.

Farahnakian et al. [4] have proposed an architecture based on multi-agent systems for dynamic VM consolidation. The authors split the problem of dynamic consolidation into two subproblems, namely host status detection and VM placement optimization. This two-level architecture uses a local agent for each host, which detects when the host is overloaded through a reinforcement learning (RL) approach. Another agent called global agent has a supervisory role. It receives information from the local agent and takes decisions for the migration of VMs.

Farahnakian et al. [5] have also proposed a VM management framework based on multi-agent systems aimed to reduce SLA violations and power consumption. The agents, arranged in a three-level hierarchical architecture, are called global, cluster and local agents. A local agent is responsible for the resource usage of the host. To coordinate the local agents by respective clusters, a cluster agent is used, and a master node runs a global agent.

Beloglazov and Buyya [11] have proposed an adaptive heuristic for dynamic consolidation of VMs.To detect when a host is overloaded or underloaded, an adaptive threshold with upper and lower bounds based on historical data is defined.

Murtazaev and Oh [6] have proposed a server consolidation algorithm called Sercon to minimize the number of used servers and number of migrations. This algorithm migrates VMs from the least loaded nodes to the most loaded ones.

In contrast to the existing dynamic VM consolidation approaches presented above, we propose a distributed resource allocation approach based on multi-agent systems. Our approach does not use static or adaptive thresholds but it is based on the utility function model based on host CPU utilization. Basing resource allocation decision on utility function optimization offers a flexible resource allocation policy that is not present in the threshold- and rule-based policies. This means that the core allocation algorithm, namely the utility function optimization mechanism, does not need to change. Changing the form of the utility function can easily change the resource allocation policy.

III. System Architecture

In this section, we discuss large-scale data center architectures consisting of *m* hosts and *n* virtual machines running on each host. Since the workload and the CPU usage change over time, an efficient approach for dynamic VM consolidation is needed.

Our work focuses on two models of architectures that are presented below. Fig. 1 depicts a two-level centralized architecture, consisting of a local agent called Host Agent and a central controller called Global Agent. The tasks of each agent are described below:

- Host Agent (HA):is responsible for continuously monitoring the host's CPU utilization and to determine whether the host is in an overloaded or underloaded state. This information is passed to the global agent that initiates live migration actions if needed. It is also responsible for initiating local allocation actions by deciding about the CPU capacity (CAP) allocation to each VM and resolving conflicts when the sum of the CAP values for all VMs is greater than the total CPU capacity.
- Global Agent (GA): makes global resource allocation decisions to optimize the VM placement in order to reduce the SLA violations and energy consumption. It gets the information from the HA for a host's status data, available CPU capacity, used CPU capacity, and the predicted CPU utilization, and performs the appropriate live migration actions.

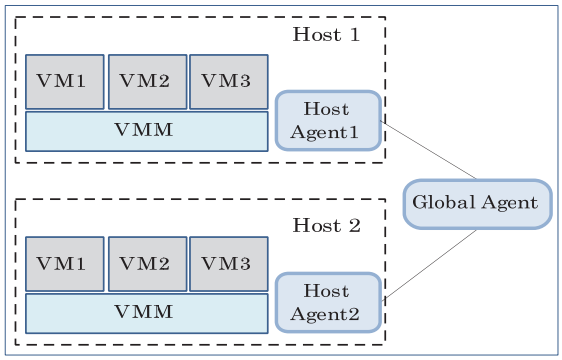

Fig. 1. Centralized allocation architecture.

Fig. 2 shows a distributed architecture where the communication is performed between the HAs.

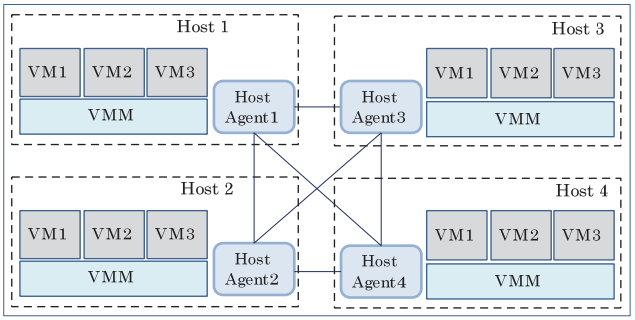

Fig. 2. Distributed allocation architecture.

In this case, a HA decides to perform live migrations without activating any central controller or GA. If the HA detects an overload or underload situation, it forwards a live migration request to another randomly selected HA to find a host as a possible destination for the VM placement.

IV. Centralized And Threshold-based Distributed Allocation Approaches

In this section, we describe a centralized resource allocation approach and a threshold-based distributed resource allocation approach.

A. Centralized Allocation

The centralized allocation (CA) approach is based on the architecture shown in Fig. 1, where the communication is performed between the GA and the HAs.The HA uses historical data to detect whether the host is in an overloaded or in an underloaded state. A host is considered as overloaded, if the actual and the past three host CPU usage valuesexceed an upper threshold. A host is considered as underloaded if the actual and the past three host CPU usage values are less than a lower threshold. If a host is in one of these situations, then the GA takes a decision for VM live migration.

In a host overload situation, the VM that has the maximum average CPU utilization is selected. After selecting the first VM, the host is checked again if it is still overloaded in order to proceed with the selection of another VM. This process continues until the host is no longer overloaded. In a host underload situation, all of its VMs are selected for migration in order to turnoff the host.

The VM placement algorithm that allocates the VMs to hosts is based on the Best First Decreasing (BFD) algorithm [12], a heuristic algorithm known for solving bin-packing problems.

B. Threshold-based Distributed Allocation

A distributed resource allocation approach is suitable for large data centers where centralized optimization complexity and a single point of failure are important factors to consider.This approach is based on the architecture shown in Fig. 2 where each HA makes live migration decisions in cooperation with other HAs.

To determine the host's overloaded or underloaded state, upper and lower thresholds are used.In this work, the lower threshold is set to 10% of the CPU capacity, and the upper threshold is calculated as the total CPU capacity (100%) minus N*5%, where N is the number of VMs.

When the HA detects a host in an overloaded or under-loaded state, it makes a live migration request to a randomly selected host, to serve as a destination for the VM placement. If the request is rejected due to insufficient resource capacity, the HA randomly tries another host. If it fails to find a suitable host, after trying a predefined number of times, it powers on a new host.The VM selection process is the same as in the centralized approach.

V. Utility-based Distributed Allocation Approach

To increase flexibility and to achieve a better overall performance in a datacenter, we propose a novel Utility-based Distributed Allocation (UDA) approach. This approach is based on the architecture shown in Fig. 2.

Unlike the DA, to make VM live migration decisions and to detect a host's overloaded or underloaded state, the UDA approach is based on a utility function model. In Fig. 3, the host utility function model used in our approach is shown, based on host CPU utilization ranging from 0 to 100%. From the graph it is evident that the best value (max of 1) of the utility function is at host CPU utilizations of 70% and 0%. The 70% of CPU utilization is optimal, since the host is fully utilized but not overloaded. The utility value is set to 1 also when the CPU utilization is 0%, with the idea to promote the removal of VMs and host shutdown when the load is low. The goal of the HA is to initiate VM live migration actions in order to maximize the utility function, resulting in optimal resource utilization.

According to the UDA approach during the monitoring process, a HA considers taking VM live migration actions if the utility value is lower than 0.8, for 4 consecutive time intervals.

In this work, the value of 4 consecutive time intervals is the average VM live migration time. It is estimated by averaging over all VM live migration times over several simulation experiments. The reason is that in order to increase the stability of the approach, live migration actions are not started immediately but only if low utility states persist longer than the migration time.

In this case, the HA should make a VM live migration request to a peer HA selected randomly.

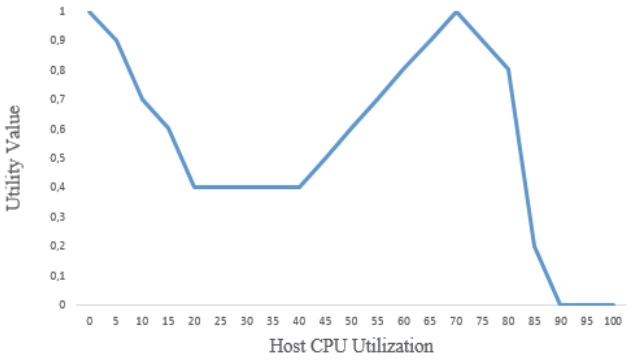

Fig. 3. The utility function model

In Fig. 4, the communication scheme between source and destination host is illustrated.

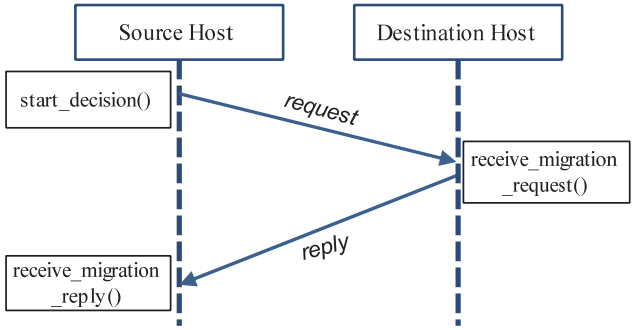

Fig. 4. Communication scheme between source and destination host.

The communication proceeds as follows. When the source HA senses that there is a low utility state, it calls the start_decision() procedure. This procedure selects a destination host randomly and sends a migration request message. The message contains all information needed by the destination HA to make migration decisions, such as a list of VMs with their CPU utilization, free RAM capacity etc. To mitigate any race conditions of receiving migration requests by other hosts, this procedure also sets the busy_with_migration variable to true.

After receiving a migration request, if busy_with_migration is false, the receiving host sets it to true and calls the receive_migration_request() procedure that is explained in Algorithm 1.

This procedure makes a migration decision of migrating one VM from source to destination host or vice versa and notifying the source host for the decision with a reply message. If the destination host is busy with processing another migration request, it sends a reply indicating the reason of request rejection as busy. The HA uses two threads, one for accepting requests and onefor making migration decisions. This is done for not blocking the sending HA for a reply. After receiving the reply, the source HA calls receive_migration_reply() that is explained in Algorithm 2.

Algorithm 1 is executed on the destination host in

response to a VM live migration request. This algorithm estimates which VM should be migrated and in which direction the maximum utility increase should be given.

Algorithm 1receive_migration_request()

```
1:  push_p = DO_NOTHING
2:  utility_before = get_utility(source_h) + get_utility(dest_h)
3:  for each VM in  source_h.VM_list do
4:    utility_after = get_utility(source_h_util -
      vm.get_avrg_util())+
    get_utility(dest_h.get_avrg_util(h) + vm.get_avrg_util());
5:  utility_increase = utility_after - utility_before;
6:      if (utility_increase>max_util_increase) &
          (dest_h.get_free_ram() >= vm.ram)  then
7:      max_util_increase = utility_increase;
8:      max_migrating_vm = vm;
9:      push_p = PUSH_VM;
10: end if
11: end for
12: for each VM in dest_h.getVMList() do
13:     utility_after = get_utility(source_h_util +
        vm.get_avrg_util())+  get_utility(dest_h.get_avrg_util(h) -
        vm.get_avrg_util());
14: utility_increase = utility_after - utility_before;
15:     if (utility_increase>max_util_increase) &
          (source_h.get_free_ram() >= vm.ram)  then
16:     max_util_increase = utility_increase;
17:     max_migrating_vm = vm;
18:     push_p = PULL_VM;
19:    end if
20: end for
21: if max_util_increase>utility_diff_thr then
22: send.Reply(push_p, migrating_vm);
23: end if
24: else
25: send.Reply(DO_NOTHING, null);
26  busy_with_migration = false;
27: end
```

Utility_before is the sum of the source and destination host utility values before VM live migration. Utility_after is the sum of the source and destination host utility values after VM live migration. The increase of the host utility value as a result of VM live migration is defined through utility_increase.

The get.avrg_util() method gives the average CPU utilization of VMs, source and destination hosts, calculated for the past 4 consecutive intervals. The utility increase should be greater than a predefined utility_diff_thr threshold in order to take a VM migration action. This is done to increase the stability of the approach and reduce unnecessary VM live migrations.

The variable busy_with_migration also is set to false on both source and destination hosts, when the VM live migration process is finished.

Algorithm 2 is executed on the source host after receiving the response from the destination host.

In Algorithm 2, the push_pull variable indicates the direction of VM live migration. If its value is DO_NOTHING, there is no live migration action because there is no increase in utility function, or the destination host is busy with another migration process.

To differentiate between the cases, the variable reject is tested to check if the destination host is busy with another migration. The busy_counter variable limits how many times to try other hosts if previous hosts are busy. The overload_counter limits how many times to try other hosts if there are no increases in utility.

Algorithm 2receive_migration_reply()

```
3:  if (push_pull == DO_NOTHING) & (reject == BUSY)then
4:     if  busy_counter != 0  then
5:  start_decision();
6:  busy_counter --;
7:  end if
8:  else
9:  busy_counter = busy_counter_thr;
10: busy_with_migration = false;
11: end
12: else ifpush_pull == DO_NOTHING  then
13: if  overload_counter != 0  then
14: start_decision();
15: overload_counter --;
16:     end if
17:    else if (source_h.get_avrg_util(h)
       >(source_h.getUpperThr(h))
18: new_h = host_power_on();
19:         migrate_vm_to_host(source_h, new_h,
            source_h.selectVM());
20: end if
21: else
22:         busy_with_migration = false;
23:         overload_counter = overload_counter_thr;
24: end
25: end if
26: else ifpush_pull == PUSH_VM   then
27: migrate_vm_to_host(source_h, dest_h, migrating_vm);
28: end if
29: else ifpush_pull == PULL_VM   then
30: migrate_vm_to_host(dest_h, source_h, migrating_vm);
31: end if
```

In both cases, the start_decision() is called to make a new migration request to another randomly selected host. If a suitable host is not found for a number of trials, because there is no increase in utility (the overload_counter variable reaches a threshold) and the source host is in an overloaded state, then a new host is powered on. In this case, selectVM() is called to select the VM that should be migrated from the source host to the new host.

If the value of the push_pull variable is PUSH_VM or PULL_VM, then it indicates a VM live migration action from source to destination host or from destination to source host, respectively.

VI. Experimental Results

In this section,we present experimental results of our proposed UDA approach compared to three other approaches. The first one, called No Migration (NOM) approach, allocates CPU resources to VMs locally, but does not perform live migration actions. The second one is the Centralized Approach (CA), and the last approach is Distributed Allocation (DA), as described in Section IV.

To have a controlled experimental environment and the possibility of repeated experimental runs, we have developed a simple event based cloud simulator. We consider datacenters of different sizes with the number of hosts varying from 100 to 700 and an initial allocation of 3 VMs per host.To simulate VM workloads, CPU usage data of real VMs running in PlanetLab [10], is used. Each VM runs one application with a variable workload. The experiment is run for 570 time intervals, and the duration of a time interval is set to 5 seconds.The simulation experiment is run for four different load levels called VLOW, LOW, HIGH and VHIGH. The load level represents the CPU usage consumed by each VM. The load

levels taken by multiplying the PlanetLab CPU usage values for each time interval with a constant value are as follows: VLOW with 0.2, LOW with 1, HIGH with 2 and VHIGH with 3. The experiment is repeated ten times for each combination of approach and load level.

We have defined four performance metrics for evaluation of the proposed approach, as discussed below.

A. VM SLA Violation (VSV)

This metric represents the penalty of the cloud provider for violating the performance of the cloud consumer VMs. The VM SLA violation happens if the difference between allocated CPU and CPU usage of the VM is less than 5% of the CPU capacity for four consecutive intervals. The reason for this metric is that the performance of an application is poor if the required CPU usage is near the allocated CPU share.

Taking into account the VM SLA violation metric above, we have defined an upper threshold for host overload detection, which is calculated dynamically depending on the number of VMs. To avoid a VM SLA violation, each host must have more than 5% capacity above the CPU usage for each VM, so the total free CPU capacity of the host should be more than N*5%, where N is the number of VMs. The overload threshold is calculated as the total CPU capacity (100%) minus N* 5%[14]. For the underload threshold, we used a fixed value of 10% of the CPU capacity.

In the final results, we show the cumulative VSV (CVSV) value that is estimated as the sum of VSV values of all VMs for the whole experimental time.

B. Energy Consumption (E)

This is an important metric,since the target of server consolidation in a datacenter is to reduce energy consumption.The total energy consumption of the datacenter, measured in KWh, for the whole experimental time is shown in the experimental results.

C. Number of VM Migrations

The process of live migration is costly because it takes a significant quantity of CPU processing on the source host, traffic load during the communication between the source and destination host [13], and can cause VM SLA violations.

D. Energy and VM SLA Violations (ESV)

This metric combines energy consumption (E) and cumulative VM SLA violation (CVSV) value in a single metric:

$$ESV = E \times CVSV(1) \qquad (1)$$

To see the effect the load level has on the VM SLA violation, Fig. 5 presents the cumulative VSV value for each approach and the four load levels. For each approach, we notice that while the load increases, the SLA violations are increased. From the results we can see that for all load levels, the UDA approach achieves the lowest CVSV value compared to the other approaches.

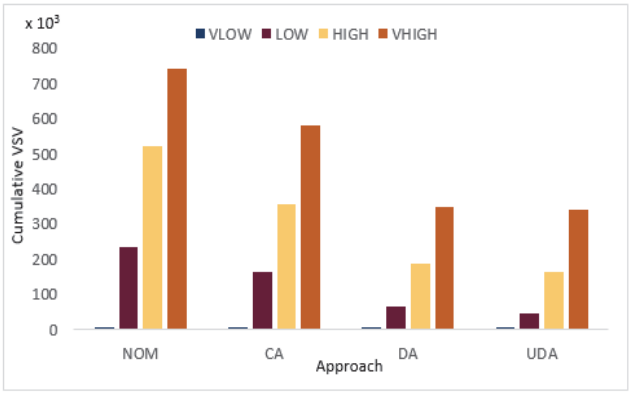

Fig. 5. Cumulative VSV over all load levels.

Fig. 6 shows the energy consumption of the data center for the whole experimental time for each approach over all load levels. It is evident that by increasing the load, the energy consumption is increased for all approaches.

The UDA approach, despite consuming more energy than the CA and DA approaches, has smaller SLA violations and therefore a lower ESV metric than other approaches, as shown in Fig. 8.

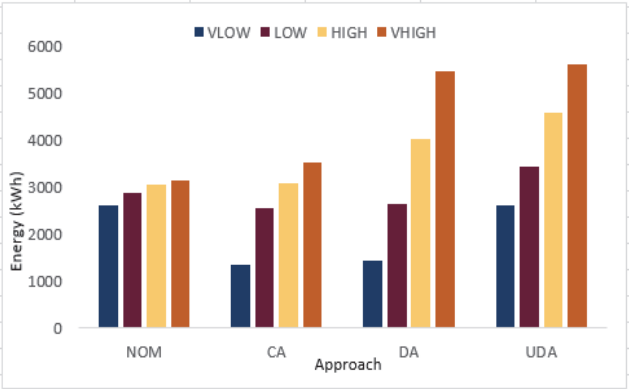

Fig. 6. Energy over all load levels.

Fig. 7 shows how the load levels affect the number of live migrations.

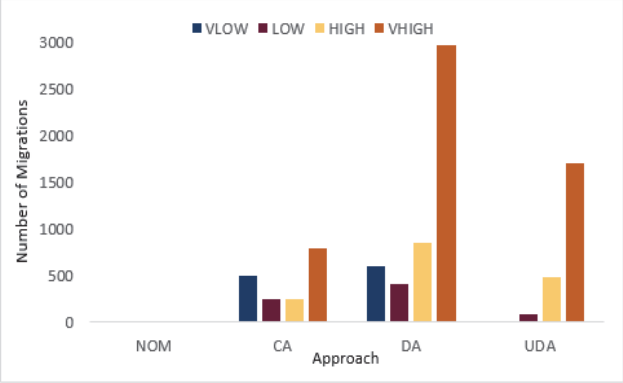

Fig. 7. Number of live migrations over all load levels.

It can be noticed that the number of live migrations of the UDA approach is significantly smaller than CA and DA for low load levels (VLOW and LOW), while for high load levels (HIGH and VHIGH), the number of live migrations of the UDA approach is smaller than DA but greater than

the CA approach. This is because for high load levels it needs more VM live migrations to achieve lower values of the SLA violations and ESV metric.

Fig. 8 shows the ESV metric over four load levels, where it is evident that the ESV metric is smaller for the UDA approach than the other approaches for each load level.

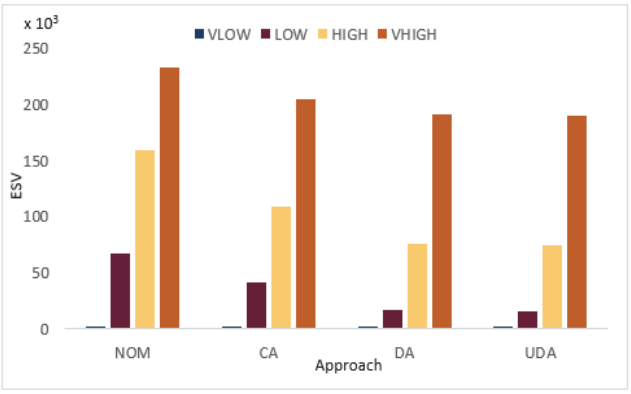

Fig. 8. ESV over all load levels.

We have also estimated the ESV metric for different number of hosts, for all approaches and four load levels. We have tested for 100, 300, 500 and 700 hosts. For space reasons, in Fig. 9 we only show the ESV value for the HIGH load level. The UDA approach achieves the smallest ESV value compared to the other approaches. The same trend of the ESV value is observed for other load levels.

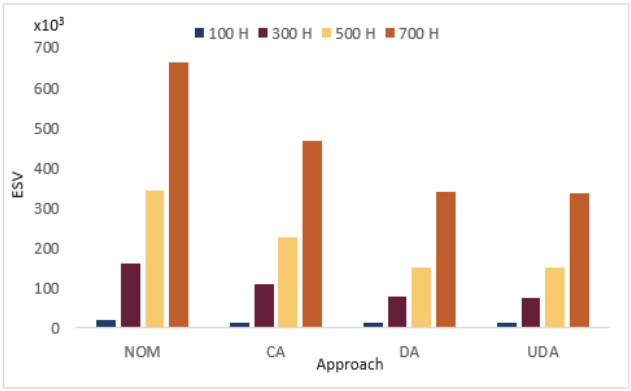

Fig. 9. ESV for HIGH load per different number of hosts.

VII. Conclusion

We have presented a multi-agent distributed approach for dynamic resource allocation in cloud infrastructures based on utility function optimization.

Compared to other approaches, the utility-based distributed resource allocation approach shows reduced VM SLA violations and minimized ESV metrics.

There are several areas for future work. First, more work is needed to select the best utility function model to improve the reduction in energy consumption. Second, the dynamic VM consolidation algorithm should take into account other resources such as memory and network I/O, since they have a significant impact on the overall performance. Finally, to further optimize VM placement, other factors such as security and network traffic should be considered.

References

[1] G. Selim, M. El-Rashidy, N. El-Fishawy, “An efficient resource utilization technique for consolidation of virtual machines in cloud computing environments”, *33rd National Radio Science Conference*, Aswan, Egypt, pp.316-324, IEEE, Feb. 22-25 2016.

[2] Q. Zhang, E. Gurses, R. Boutaba, J. Xiao, ”Dynamic resource allocation for spot markets in cloud computing environments”, *Fourth IEEE International Conference Utility and Cloud Computing*, pp. 178-185, IEEE , 5-8 Dec, 2011.

[3] Brendan Jennings and Rolf Stadler, ”Resource management in clouds: survey and research challenges”, *Journal of Network and Systems Management*, Volume 23, Issue 3, pp. 567–619, Springer, July 2015.

[4] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, H. Tenhunen, ”Multi-Agent based architecture for dynamic VM consolidation in cloud data centers”, *40th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 111–118, IEEE, August 2014.

[5] F. Farahnakian, P. Liljeberg, T. Pahikkala, J. Plosila and H. Tenhunen, ”Hierarchical VM management architecture for cloud data centers”, *IEEE 6th International Conference onCloud Computing Technology and Science*,pp. 306-311, IEEE, 2014.

[6] A. Murtazaev and S. Oh, “Sercon: Server consolidation algorithm usinglive migration of virtual machines for green computing”, *Journal: IETE Technical Review*, Volume 28, Issue 3, pp. 212-231, Taylor & Francis,2011.

[7] E. Feller, C. Morin, and A. Esnault, ”A case for fully decentralized dynamic VM consolidation in clouds”, *Cloud Computing Technology and Science (CloudCom),4th IEEE International Conference*, pp. 26-33, IEEE, December 2012.

[8] M.Marzolla, O.Babaoglu and F.Panzieri, ”Server consolidation in clouds through gossiping”, *World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE International Symposium*, IEEE, 20-24 June 2011.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose and R. Buyya, ”CloudSim: a toolkit for modeling and simulation of cloudcomputing environments and evaluation of resource provisioning algorithms”, *Software - Practice and Experience*, vol. 41, pp. 23–50, ACM, 2011.

[10] K. Park and V. S. Pai, ” CoMon: A mostly-scalable monitoring system for PlanetLab”, *ACM SIGOPS Operating Systems Review*, vol. 40, Issue 1, pp. 65-74, ACM, 2006.

[11] A. Beloglazov and R. Buyya, “Optimal online deterministic algorithmsand adaptive heuristics for energy and performance efficient dynamicconsolidation of virtual machines in cloud data centers”, *Journal Concurrency and Computation: Practice and Experience*, vol. 24, issue. 13, pp. 1397–1420, ACM, 2012.

[12] M. A. H. Monil and R. M. Rahman,”VM consolidation approach based on heuristics, fuzzy logic, and migration control”, *Journal of Cloud Computing: Advances, Systems and Applications*, DOI 10.1186/s13677-016-0059-7, Springer, 2016.

[13] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, and H. Tenhunen, ” Energy-aware VM Consolidation in Cloud DataCenters Using Utilization Prediction Model”, *IEEE Transaction on Cloud Computing*, Vol. XX, No. X, IEEE, 2016.

[14] D. Minarolli, A.Mazrekaj and B. Freisleben, ”Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing”, *Journal of Cloud Computing: Advances, Systems and Applications*, DOI 10.1186/s13677-017-0074-3, Springer, 2017.

[15] T. Wood, P. Shenoy, A. Venkataramani and M. Yousif, ”Sandpiper: black-box and gray-box resource management for virtual machines”, *Computer Networks*, vol. 53, Issue 17, pp. 2923–2938, ELSEIVER, December 2009.

[16] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The eucalyptus open-source cloud-computing system”, *Cluster Computing and the Grid, CCGRID '09. 9th IEEE/ACM International Symposium*, pp. 124–131, IEEE/ACM, 2009.

[17] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, ”It's not easy being green”, *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 211-222, ACM, 2012.