A Novel Physical Machine Overload Detection Algorithm Combined with Quiescing for Dynamic Virtual Machine Consolidation in Cloud Data Centers

Loiy Alsbatin[1], Gürcü Öz[1], and Ali Ulusoy[2]
[1]Department of Computer Engineering, Eastern Mediterranean University, North Cyprus via Mersin 10 Turkey
[2]Department of Information Technology, Eastern Mediterranean University, North Cyprus via Mersin 10 Turkey

Abstract: *Further growth of computing performance has been started to be limited due to increasing energy consumption of cloud data centers. Therefore, it is important to pay attention to the resource management. Dynamic virtual machines consolidation is a successful approach to improve the utilization of resources and energy efficiency in cloud environments. Consequently, optimizing the online energy-performance trade off directly influences Quality of Service (QoS). In this paper, a novel approach known as Percentage of Overload Time Fraction Threshold (POTFT) is proposed that decides to migrate a Virtual Machine (VM) if the current Overload Time Fraction (OTF) value of Physical Machine (PM) exceeds the defined percentage of maximum allowed OTF value to avoid exceeding the maximum allowed resulting OTF value after a decision of VM migration or during VM migration. The proposed POTFT algorithm is also combined with VM quiescing to maximize the time until migration, while meeting QoS goal. A number of benchmark PM overload detection algorithms is implemented using different parameters to compare with POTFT with and without VM quiescing. We evaluate the algorithms through simulations with real world workload traces and results show that the proposed approaches outperform the benchmark PM overload detection algorithms. The results also show that proposed approaches lead to better time until migration by keeping average resulting OTF values less than allowed values. Moreover, POTFT algorithm with VM quiescing is able to minimize number of migrations according to QoS requirements and meet OTF constraint with a few quiescings*.

Keywords: *Distributed systems, cloud computing, dynamic consolidation, overload detection and energy efficiency.*

Received November 15 2017; accepted July 29, 2018
https://doi.org/10.34028/iajit/17/3/9

1. Introduction

Cloud environment is an efficient solution for data intensive and computing intensive applications [10]. Cloud computing provides scalable virtualized resources to global users over the internet. The designers of cloud computing systems have interested on the improvements of computing performance that are driven by the demand of consumer, business, and scientific applications. However, the power consumption of USA data centers has increased by 62.5% from 2005 to 2013 and expected to increase by 150% in 2020 [33]. Most of the energy consumption of data centers is consumed by the computing resources. Accordingly, it is important to pay attention to the resource management ensuring that the applications efficiently utilize the available computing resources. Switching the idle nodes to sleep mode to eliminate the idle power consumption can achieve a reduction in energy consumption. Dynamic Virtual Machine (VM) consolidation can effectively improve the utilization of resources and reduces energy consumption in data centers. Reallocating VMs from an overloaded Physical Machine (PM) maximizes the utilization and energy efficiency with providing a high Quality of Service (QoS). The goal of consolidation of VMs ensuring an efficient utilization can be achieved through the use of VMs migration across different PMs.

One efficient way to improve the utilization of cloud data center resources is the dynamic consolidation of VMs [1, 2, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15, 18, 20, 23, 30, 31, 34, 35]. The dynamic consolidation reallocates VMs periodically using migration to reduce the number of active PMs required to handle requests. The idle PMs are switched to sleep modes with fast transition times to minimize the overall energy consumption. If the demand for resources increases, PMs are reactivated from the sleep mode. The objective of this approach is mainly to minimize energy consumption and maximize of QoS.

It is complex to solve dynamic VM consolidation problem analytically as a whole [15, 30]. In general, the problem can be decomposed into tasks as following [4]:

- PM underload detection: It is the phase when a PM is considered as being underloaded, so all VMs running on an underloaded PM should be migrated to other PMs and the underloaded PM should be switched to the sleep mode (to reduce the number of active PMs).
- PM overload detection: It is the phase when a PM is considered as being overloaded, so some VMs running on an overloaded PM should be migrated to another active PM (to avoid violation QoS requirements).
- VM selection: It is the phase to select VMs to be migrated from the overloaded PM.
- VM migration: It is the phase to perform VM migration with minimal service downtime during the migration process.
- VM placement: It is the phase to place selected VMs for migration on another active PM.

In this paper, we mainly focus on PM overload detection problem. PM overload detection directly influences QoS, because performance degradation is very likely to occur if the resources are completely utilized. PM overload detection problem is complex because of the need to improve the system response time, while handling a set of heterogeneous workloads placed on a single PM [4]. When a PM is considered as being underloaded, its VMs are consolidated into other active PMs and it should be switched to the sleep mode. But when a PM is considered as being overloaded, VMs require a higher performance and PM in sleep mode is reactivated to migrate VMs.

The rest of the paper is organized as follows. In section 2, we discuss the related work. We introduce the dynamic consolidation of VMs and propose our PM overload detection algorithm and the combination of PM overload detection algorithm and VM quiescing in section 3. In section 4, we introduce the experimental settings of Central Processing Unit (CPU) model. In section 5, the experimental evaluations and results are discussed. Finally, we conclude the results and discuss the future work in section 6.

2. Related Work

In the past few years, many approaches to the dynamic consolidation of VMs have been proposed VMs [1, 2, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15, 18, 20, 23, 30, 31, 34, 35]. Some of VM consolidation algorithms based on different heuristics on the legitimate PM were analyzed by Kaushar *et al*. [17]. A comparative study of various existing consolidation of VMs algorithms using real world workload traces was presented by authors. VM consolidation algorithms under QoS expectations were evaluated using the CloudSim [5] toolkit showing high improvement of cost savings and energy efficiency using dynamic workload scenarios. A scheduling algorithm to assign VMs to PMs in a data center was proposed by Sharifi *et al*. [26]. The goal was to improve energy efficiency by taking into consideration the conflicts between the costs of VM migration and CPU and disk utilizations. Four models were presented to identify the conflicts, namely the migration model, the energy model, the application model, and the target system model.

An adaptive threshold-based algorithm was proposed by Deng *et al*. [7]. The overload threshold of CPU utilization and the average utilization of active PMs were used for PM underload detection algorithm, and minimum average utilization difference of the data center was used for VM placement algorithm. Several dynamic VM consolidation algorithms were proposed by Khoshkholghi *et al*. [19] to improve the utilization, energy consumption and Service Level Agreement (SLA) violations based on the CPU, Random Access Memory (RAM) and bandwidth. They used an iterative weighted linear regression method for PM overload detection and a vector magnitude squared of resources for PM underload detection. They also proposed SLA and power-aware VM selection algorithm and VM placement algorithm. PM overload and underload detection algorithms based on dynamic thresholds were proposed by Najari *et al*. [22]. They used simple exponential smoothing technique to predict CPU utilization and calculate dynamic upper and lower utilization thresholds. A VM consolidation algorithm with utilization prediction of multiple resource types based on the local history of PMs was proposed by Nguyen *et al*. [24] to improve the energy efficiency of cloud data centers.

Managing resource allocation to improve response time using control loops at the server and cluster levels were applied by Wang *et al*. [32]. The server migrated a VM if the server's resource capacity was not enough to meet SLAs of application. Kakadia *et al*. [16] proposed a greedy consolidation algorithm based on VMs placement algorithm to improve the network usage and performance of applications in the data centers. The greedy consolidation algorithm reduced the number of migrations and speed up the placement decisions. Forsman *et al*. [8] proposed two algorithms, which could be used together for live migration of multiple VMs. The proposed VM migration depended on three factors that were the cost of migration, the expected distribution of workload and the state of PM after migration. The algorithms distributed the workload efficiently in the system. In spite of that, the paper did not discuss how to meet SLA. A dynamic consolidation of VMs for web applications was implemented by Guenter *et al*. [13]. The response time was used to define SLAs. Weighted linear regression was applied to get the future workload and improve the distribution of workload.

Because of constraints on the allowed number of migration of VMs in a certain period of time, the prior works might not be the only effective methods. We propose an approach that uses the combination of PM overload detection algorithm and VM quiescing which maximizes the time between sequential VM migrations (to minimize the number of migration of VMs) under the specified QoS constraint.

3. Dynamic Consolidation of Virtual Machines

In general, current solutions to PM overload detection problem are based on heuristic or statistical analysis of historical data. These approaches do not clearly specify QoS target. We propose the dynamic consolidation of VMs under the specified QoS targets based on a combination of PM overload detection algorithm and VM quiescing which solves optimally the problem of short time intervals between sequential VM migrations under QoS goal. The average time between sequential VM migrations is necessary to be maximized to optimize consolidation of VMs [4]. VM consolidation is used to reduce the number of active PMs, so a lower average number of active PMs represents a better consolidation of VMs.

PM overload detection can initiate a VM migration from an overloaded PM. There are two possible cases of VM migrations due to the overload:

1. A new PM must be activated to a VM, which should be migrated from an overloaded PM due to not enough resources on another active PM.
2. A VM which should be migrated can be placed on another active PM.

The goal of PM overload detection is to maximize the average time between sequential VM migrations (to minimize the number of migrations) [4]. Therefore, the activity time should be maximized for overloaded PMs, while the activity time should be minimized for under loaded PMs.

At every moment of time, each VM on a PM takes a fraction of CPU utilization as required by VM's workload. PM's workload is constituted by CPU utilization created by a set of VMs that is currently allocated to a PM. CPU utilization of PM is monitored by a controller. A PM overload detection algorithm decides when a VM migration needs to be done to meet QoS goals, while maximizing the average time until migration.

3.1. A Workload QoS Metric

We use a workload QoS metric used in [4] to impose QoS requirements on the system. PM can be in one of the states according to its CPU utilization:

1. Serving the normal load.
2. Being highly loaded.

It is assumed that VMs allocated to the overloaded PM might not provide the required performance level. Therefore, this leads to performance degradation. We use Overload Time Fraction (OTF) metric which allows measuring the performance degradation over the interval of time in regard to the definition of the overload state [4]. OTF metric is defined as:

$$\mathrm{OTF}(100\%) = \frac{t_o(100\%)}{t_a} \quad (1)$$

Where t_o is the total time during which PM is in the overload state, when its CPU utilization is 100%, and t_a is the total time of PM being in the active state. OTF is monitored continuously, and is recalculated every time PM overload detection is invoked.

3.2. Proposed PM Overload Detection Algorithm

We propose PM overload detection algorithm based on OTF Threshold (OTFT) algorithm [4]. OTFT algorithm decides to migrate a VM if the current OTF value of PM exceeds the defined Maximum Allowed OTF (MAOTF) value. However, OTFT algorithm fails to meet SLA requirements because OTF threshold equals to MAOTF and the average resulting OTF value exceeds the maximum allowed resulting OTF value [4].

Our proposed overload detection algorithm decides to migrate a VM if the current OTF value of PM (pm.OTF) exceeds the defined percentage (p) of MAOTF value to avoid exceeding the maximum allowed resulting OTF value after a decision of VM migration or during VM migration. We refer to this algorithm as Percentage of Overload Time Fraction Threshold (POTFT) algorithm. POTFT algorithm is presented in Algorithm 1.

Algorithm 1: POTFT PM Overload Detection Algorithm

```
Input: pm.OTF, MAOTF, p
Output: PM's status
1: PM's status = normal loaded
2:      if pm.OTF > (MAOTF × p) then
3:          PM's status = overloaded
4:      end if
5: return PM's status
```

3.3. Proposed PM Overload Detection Algorithm with VM Quiescing

Current researches may not be able to limit the number of VM migrations to be less than maximum allowed number of VM migrations and at the same time do not allow PM to exceed maximum allowed OTF. Therefore, this paper focuses on PM overload detection algorithms and proposes the combination of proposed PM overload detection algorithms with VM quiescing (temporary turning off VM) [3], which is able to minimize number of VM migrations and meet OTF constraint. VM quiescing should be applied with

efficient PM overload detection algorithm to minimize number of VM quiescings required. Therefore, we propose to combine VM quiescing with POTFT algorithm to minimize the number of VM migrations to be less than maximum allowed number of VM migrations and limit OTF to be less than MAOTF according to QoS constraints.

PM overload detection algorithm is invoked periodically to decide if a PM is overloaded or not. The proposed algorithm turns off one of VMs randomly from the set of generated VM by Algorithm 3 if PM is highly loaded according to POTFT algorithm that its OTF value exceeds specified percentage of MAOTF, and current simulation time does not exceed minimum allowed time until migration. Turned off VMs are restarted when PM no longer suffers from overload. PM is no longer considered suffering from overload when OTF value of PM is less than 10% subtracted from specified percentage of MAOTF value. If PM still suffers from overload and current simulation time exceeds minimum allowed time until migration, a VM should be migrated from PM. The maximum allowed number of VM migrations initiated over n time steps (n/T) is considered as QoS requirement, where T is the minimum allowed time until migration and is set according to the maximum allowed number of VM migration in time. Combination of POTFT and VM quiescing algorithm is presented in Algorithm 2.

Algorithm 2: Combination of POTFT and VM Quiescing Algorithm

Input: A set of generated VMs, a set of turned off VMs, pm.OTF, T, current simulation time, MAOTF, p
Output: A set of turned off VMs and a decision on whether to migrate a VM
1: a decision on whether to migrate a VM = false
2: if pm.OTF > (MAOTF × p) then
3: if current simulation time > T then
4: a decision on whether to migrate a VM = true
5: else VM Quiescing
6: select VM randomly from a set of generated VMs
7: remove CPU utilization trace of a selected VM
8: add a VM to the set of turned off VMs
9: end if
10: else
11: if the set of turned off VMs ≠ null and pm.OTF < (MAOTF × (p - 0.1)) VM Unquiescing
12: select a VM randomly from turned off VMs
13: assign CPU utilization trace for the selected VM
14: remove a VM from the set of turned off VMs
15: end if
16: end if
17: return (a set of turned off VMs, a decision on whether to migrate a VM)

3.4. Benchmark PM Overload Detection Algorithms

We evaluate the proposed algorithms using a number of benchmark PM overload detection algorithms with different parameters. The first benchmark algorithm is a simple heuristic algorithm based on specifying fixed CPU utilization Threshold (THR), which is applied in a number of related works [11, 12, 31, 35]. PM's CPU utilization is monitored and if the specified upper threshold is exceeded, a VM is migrated. The next two algorithms are based on the statistical analysis to adjust CPU utilization threshold dynamically: based on Median Absolute Deviation (MAD) and Interquartile Range (IQR) [5]. MAD adjusts upper and lower PM's utilization thresholds and keeps the total utilization of VMs between these thresholds. IQR adjusts an upper PM's utilization threshold based on a measure of statistical dispersion, being equal to the first quartile subtracted from the third quartile. The next two algorithms estimate the future CPU utilization using regression-based approach and a modification of the robust regression method, which is robust to outliers [5]. These algorithms are denoted as Local Regression (LR) and Local Regression Robust (LRR) respectively. The main idea of LR, which is proposed by Guenter *et al*. [13], is to fit simple models to localized subsets of data to build up a curve that approximates the original data. LR algorithm is in line with robust regression method. Two other algorithms discussed in section 3.2. are OTFT and our proposed POTFT algorithms. In this paper, the benchmark PM overload detection algorithms are compared to proposed POTFT algorithm and POTFT with VM quiescing as presented in section 5.2.

4. CPU Modelling

We use CPU model as presented in [4]. The model is appropriate for single core and multi-core CPU architectures. The single core CPU capacity is modelled according to its clock frequency (F). A CPU utilization of VMs (u_i) is a fraction of CPU utilization of PM (U) and is relative to CPU frequency of VMs (f_i). CPU utilization of PM equals the summation of fractions of L VMs running on PM as:

$$U = \frac{1}{F}\sum_{i=1}^{L} f_i u_i \qquad (2)$$

We model a multi-core CPU for the investigation of PM overload detection problem. A clock frequency (F_c) of a multi-core CPU with k cores is modelled as kF_c frequency of a single core CPU, which means F in (2) is replaced by kF_c. Using a time-shared scheduling algorithm, each VM is randomly assigned to one of CPU's cores. The only restriction is that VM's CPU capacity cannot exceed the single core capacity. If this restriction is removed, a VM would be required to be executed on multi-core in parallel.

5. Performance Evaluation of PM Overload Detection Algorithms using CPU Model

We use simulations to evaluate the proposed POTFT algorithm and POTFT with VM quiescing. We use the source code of benchmark PM overload detection

algorithms which is written in Clojure [4] and add the code of proposed POTFT algorithm and VM quiescing algorithm.

5.1. Evaluation of PM Overload Detection Algorithms using Planet-Lab Workload Traces

In multiple PMs environment, PM overload detection algorithm works independently in a decentralized way on every PM. So, a variety of heterogeneous VMs is served using a single PM with a quad-core CPU to evaluate PM overload detection algorithm under a real world workload. The clock frequency of each core of PM is set to 3 GHz, which transforms into 12 GHz according to CPU model in Section 4. The above CPU characteristics correspond to a medium range type in the cloud physical Amazon EC2 servers [21]. The memory size of PM is assumed to be enough for VMs. CPU frequency of each created VM is randomly set to: 1.7 GHz, 2 GHz, 2.4 GHz, or 3 GHz, which corresponds to the types of Amazon EC2 instance [27]. CPU utilization used in the simulations is based on workload traces from the CoMon project, a monitoring tool for Planet-Lab [25]. The provided workload traces were collected every 5 minutes from more than a thousand VMs located at more than 500 places around the world. This trace was taken from March to April in 2011.

To run a simulation, a set of VMs is generated randomly with the assigned CPU utilization traces allocated on PM. PM overload detection technique at each time step decides if a VM migration should be done or not. The simulation ends when a VM is decided to be migrated, or when all workload traces are assigned. When a simulation ends, the average OTF is calculated according to (1). A set of VMs is assigned with the workload traces by the workload trace assignment algorithm which is presented in Algorithm 3 [4]. The original workload traces is filtered to assign more dynamic workloads to PM overload detection algorithms. MAOTF after the first 30 time steps is constrained to 10% and the minimum overall OTF to 20%. The workload trace assignment algorithm regenerates 100 different sets of VMs that meet the specified OTF constraints and every PM overload detection algorithm is run for each set of VMs [4].

Algorithm 3: Workload Trace Assignment Algorithm

Input: A set of CPU utilization traces
Output: A set of generated VMs
1: select PM's minimum CPU utilization randomly from (80%, 85%, 90%, 95%, and 100%) at the time 0
2: while PM's CPU utilization < PM's minimum CPU utilization at the time 0
3: randomly select CPU frequency of new VM
4: randomly assign a CPU utilization trace
5: add new VM to the set of created VMs
6: end while
7: return a set of generated VMs

The output of Algorithm 3 is used as one of the inputs to Algorithm 2.

5.2. Simulation Results

In this section, we compare THR, MAD, IQR, LR, LRR, OTFT, POTFT, and POTFT with VM quiescing algorithms with the experimental environment settings presented in section 5.1. For each overload detection technique, the parameters are used as presented in Table 1 [4].

Table 1. Parameters used in PM overload detection algorithms.

Algorithm	Parameter	Value 1	Value 2	Value 3
THR	CPU utilization threshold [4, 5, 9, 11]	80%	90%	100%
MAD	The median of the absolute deviations [12]	2	3	-
IQR	The median of the absolute deviations [11]	1	2	-
LR	Estimated trend line [12]	1.2	1.1	1.0
LRR				
OTFT	OTF threshold (MAOTF) [4]	10%	20%	30%
POTFT	OTF threshold (MAOTF × *p*)	(10%×80%) 8%	(20%×80%) 16%	(30%×80%) 24%
		(10%×85%) 8.5%	(20%×85%) 17%	(30%×85%) 25.5%
		(10%×90%) 9%	(20%×90%) 18%	(30%×90%) 27%

MAOTF value of OTFT and POTFT is set to 10%, 20% and 30%. OTF threshold for OTFT is set as equal to MAOTF (10%, 20% and 30%) as presented in [4]. And OTF threshold for POTFT is set to 8%, 8.5%, 9%, 16%, 17%, 18%, 24%, 25.5%, and 27% with varied *p* (80%, 85% and 90%) multiplied by MAOTF (10%, 20% and 30%). Average value of resulted OTF was not exceeding 10%, 20% and 30% respectively, by tuning parameters of PM overload detection algorithm to be increased from first to third parameter as shown in Table 1. Minimum allowed time until migration (*T*) is set to 40,000 seconds and 80,000 seconds for all algorithms. OTF parameter values that are monitored continuously and recalculated every time PM overload detection is invoked as stated in section 3.1 and the minimum allowed time until migration are varied according to various supposed QoS constraints. 43 various combinations of the algorithms and parameters result by these variations. VM quiescing may occur many times before migration. The better algorithm is the one that maximizes the time until migration and does not let resulting OTF value to exceed maximum allowed value, which satisfies QoS constraint.

Figure 1 presents the average OTF values with 95% confidence interval for benchmark PM overload detection algorithms, proposed POTFT without

quiescing, and proposed POTFT with VM quiescing (POTFT_Q). It can be seen from the results in Figure 1 that POTFT and POTFT_Q with all parameters are the only competitive algorithms. They maximize resulting OTF value without allowing it to exceed maximum allowed value (10%, 20%, and 30%) and maximize the time until migration, while POTFT_Q is able to maximize time until migration to be more than minimum allowed time until migration as shown in Figure 2.

Figure 2 presents the average time until a migration with 95% confidence interval for PM overload detection algorithms. The results in Figure 3 show that OTFT and proposed POTFT without quiescing have longer time until migration compared to other overload detection algorithms, and POTFT_Q is able to maximize time until migration to be more than minimum allowed time until migration (40,000 seconds and 80,000 seconds). The results in Figure 3 show that there is a statistically significant difference in the average time until a migration produced by LR_1, LRR_1, OTFT_30, POTFT_24, POTFT_25.5, and POTFT_27 algorithms compared to other algorithms except POTFT_Q. OTFT and POTFT algorithms have better average time until a migration compared to LR, and LRR. Moreover, the resulting OTF of POTFT and POTFT_Q and the time until migration of POTFT are increased when *p* parameter is increased from 80% to 90%.

Figure 3 presents the average number of VM quiescings with 95% confidence interval for POTFT_Q to maximize the time until migration to 40,000 seconds and 80,000 seconds. A large number of quiescings may highly affect QoS. However, Figure 3 shows that POTFT_Q with varied parameters has a small number of quiescings. Therefore, POTFT_Q is able to maximize time until migration according to QoS requirements with a few quiescings.

Table 2. SLA violations by OTFT and POTFT.

OTF Parameter	OTFT	POTFT_80%	POTFT_85%	POTFT_90%
10%	100/100	0/300	0/300	3/300
20%	100/100	0/300	0/300	0/300
30%	44/100	0/300	0/300	0/300
Overall	81.33%	0%	0%	0.33%

Table 2 presents the levels of SLA violations caused by OTFT and POTFT algorithms. The results show that POTFT significantly outperforms OTFT algorithm according to SLA violation levels. POTFT with 80% and 85% of MAOTF leads to 0% SLA violations, POTFT with 90% of MAOTF causes only 0.33% SLA violations, whereas OTFT causes 81.33% SLA violations.

The results in Figure 2 show that the average time until a migration of POTFT is slightly lower than OTFT algorithm, however POTFT is able to avoid SLA violations as shown in Table 2. The experimental results show that the proposed POTFT leads to higher average time until a migration compared to benchmark algorithms while meeting the specified OTF goal. Moreover, POTFT_Q is able to minimize number of migrations according to QoS requirements and meet OTF constraint with a few quiescings.

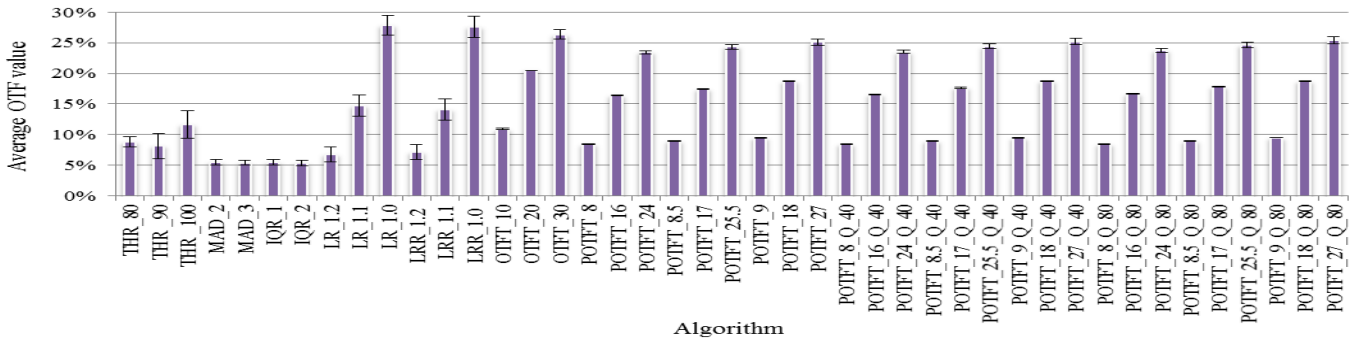

Figure 1. Average OTF values with 95% confidence interval of PM overload detection algorithms.

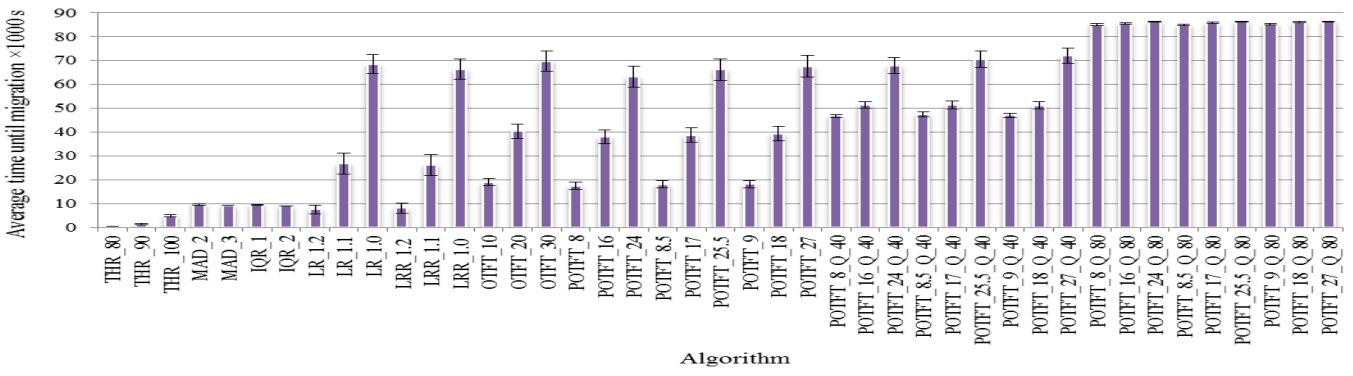

Figure 2. Average time until a VM migration with 95% confidence interval of PM overload detection algorithms.

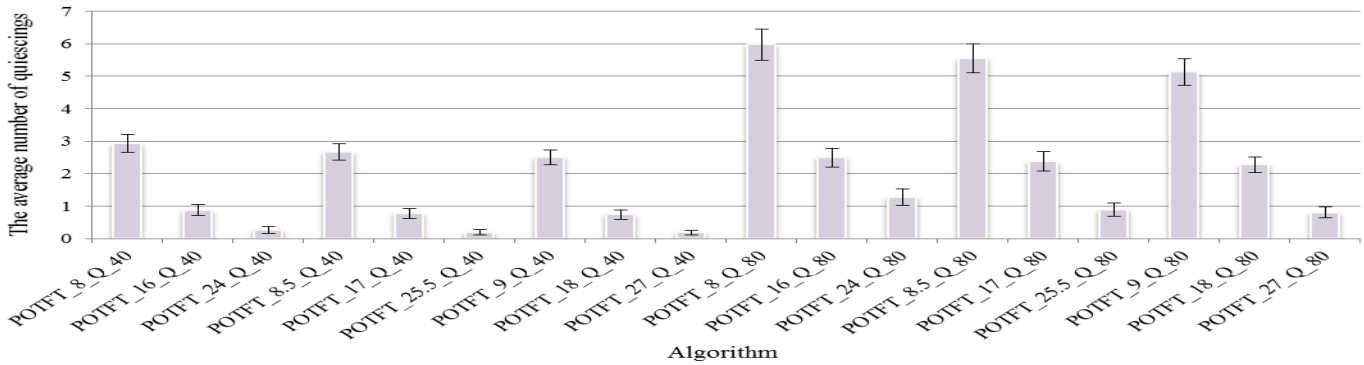

Figure 3. Average number of quiescings with 95% confidence interval of combination of POTFT and VM quiescing.

6. Conclusions

Current works may not be able to limit the number of VM migrations to be less than maximum allowed number of VM migrations and at the same time do not allow PM to exceed MAOTF according to QoS constraints. Therefore, we propose dynamic VM consolidation based on a PM overload detection algorithm and a combination of PM overload detection algorithm and VM quiescing to minimize number of VM migrations according to QoS requirements and meet OTF constraint. The goal of the model is to improve the utilization of resources and energy efficiency in cloud data centers. We implement a number of PM overload detection algorithms using various parameters to compare with the proposed POTFT with and without VM quiescing. The algorithms are evaluated through simulations using real world workload traces. Our results of experiments show that PM overload detection algorithm outperforms the benchmark PM overload detection algorithms and leads to higher performance of benchmark PM overload detection algorithms, while meeting QoS target. The results show that proposed PM overload detection algorithm leads to better time until migration keeping average resulting OTF values less than allowed values. Moreover, proposed POTFT with VM quiescing is able to minimize number of VM migrations according to QoS requirements and meet OTF constraint with a few quiescings.

As a future work, we plan to implement the proposed combination of VM quiescing and benchmark PM overload detection algorithms on a software framework for dynamic and energy efficient consolidation of VMs called OpenStack Neat [28]. The framework can be applied in existing OpenStack Clouds [29] deployments and in research on dynamic consolidation of VMs to optimize the resource utilization and energy efficiency.

References

[1] Alsbatin L., Oz G., and Ulusoy A., "An Overview of Energy-Efficient Cloud Data Centers," *in Proceedings of the International Conference of Computer and Applications*, Dubai, pp. 211-214, 2017.

[2] Arianyan E., Taheri H., Sharifian S., and Tarighi M., "New Six-Phase On-line Resource Management Process for Energy and SLA Efficient Consolidation in Cloud Data Centers," *The International Arab Journal of Information Technology*, vol. 15, no. 1, pp. 10-20, 2018.

[3] Baset S., Wang L., and Tang C., "Towards an Understanding of Oversubscription in Cloud," *in Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Berkeley, pp. 1-6, 2012.

[4] Beloglazov A. and Buyya R., "Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers Under Quality of Service Constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1366-1379, 2013.

[5] Beloglazov A. and Buyya R., "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," *Concurrency and Computation: Practice and Experience*, vol. 24, pp. 1397-1420, 2012.

[6] Chunlin L., Yanpei L., and Youlong L., "Energy-Aware Cross-Layer Resource Allocation in Mobile Cloud," *International Journal of Communication Systems*, vol. 30, no. 12, pp. e3258-n/a, 2017.

[7] Deng D., He K., and Chen Y., "Dynamic Virtual Machine Consolidation for Improving Energy Efficiency in Cloud Data Centers," *in Proceedings of 4th International Conference on Cloud Computing and Intelligence Systems*, Beijing, pp. 366-370, 2016.

[8] Forsman M., Glad A., Lundberg L., and Ilie D., "Algorithms for Automated Live Migration of Virtual Machines," *Journal of System and Software*, vol. 101, pp. 110-126, 2015.

[9] Fu X. and Zhou C., "Virtual Machine Selection and Placement for Dynamic Consolidation in

Cloud Computing Environment," *Frontiers of Computer Science*, vol. 9, no. 2, pp. 322-330, 2015.

[10] Gao K., Wang Q., and Xi L., "Reduct Algorithm Based Execution Times Prediction in Knowledge Discovery Cloud Computing Environment," *The International Arab Journal of Information Technology*, vol. 11, no. 3, pp. 268-275, 2014.

[11] Gmach D., and Rolia J., Cherkasova L., Belrose G., Turicchi T., and Kemper A., "An integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics," *in Proceedings of the 38th IEEE International Conference on Dependable Systems and Networks*, Anchorage, pp. 326-335, 2008.

[12] Gmach D., Rolia J., Cherkasova L., and Kemper A., "Resource Pool Management: Reactive Versus Proactive or Lets Be Friends," *Computer Networks*, vol. 53, no. 17, pp. 2905-2922, 2009.

[13] Guenter B., Jain N., and Williams C., "Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning," *in Proceedings of the 30st Annual IEEE Intl. Conference on Computer Communications*, Shanghai, pp. 1332-1340, 2011.

[14] Han G., Que W., Jia G., and Shu L., "An Efficient Virtual Machine Consolidation Scheme for Multimedia Cloud Computing," *Sensors*, vol. 16, no. 2, pp. 246-246, 2016.

[15] Jung G., Hiltunen M., Joshi K., Schlichting R., and Pu C., "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," *in Proceedings of the 30th Intl. Conf. on Distributed Computing Systems*, Genova, pp. 62-73, 2010.

[16] Kakadia D., Kopri N., and Varma V., "Network-Aware Virtual Machine Consolidation for Large Data Centers," *in Proceedings of the 3rd International Workshop on Network-Aware Data Management*, Denver, pp. 1-8, 2013.

[17] Kaushar H., Ricchariya P., and Motwani A., "Comparison of SLA based Energy Efficient Dynamic Virtual Vachine Consolidation Algorithms," *International Journal of Computer Applications*, vol. 102, no.16, pp. 31-36, 2014.

[18] Khan M., Paplinski A., Khan A., Murshed M., and Buyya R., *Sustainable Cloud and Energy Services*, Springer, 2018.

[19] Khoshkholghi M., Derahman M., Abdullah A., Subramaniam S., and Othman M., "Energy-Efficient Algorithms for Dynamic Virtual Machine Consolidation in Cloud Data Centers," *IEEE Access*, vol. 5, pp. 10709-10722, 2017.

[20] Kumar S., Talwar V., Kumar V., Ranganathan P., and Schwan K., "vManage: Loosely Coupled Platform And Virtualization Management in Data Centers," *in Proceedings of the 6th International Conference on Autonomic Computing*, Barcelona, pp. 127-136, 2009.

[21] Mills K., Filliben J., and Dabrowski C., "Comparing vm-Placement Algorithms for on-Demand Clouds," *in Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Athens, pp. 91-98, 2011.

[22] Najari A., Alavi S., and Noorimehr M., "Optimization of Dynamic Virtual Machine Consolidation in Cloud Computing Data Centers," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 9, pp. 202-208, 2016.

[23] Nathuji R. and Schwan K., "Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 265-278, 2007.

[24] Nguyen T., Francesco M., and Yla-Jaaski A., "Virtual Machine Consolidation with Multiple Usage Prediction for Energy-Efficient Cloud Data Centers," *IEEE Transactions on Services Computing*, vol. 99, pp. 1-14, 2017.

[25] Park K. and Pai V., "CoMon: A Mostly-Scalable Monitoring System for Planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65-74, 2006.

[26] Sharifi M., Salimi H., and Najafzadeh M., "Power-Efficient Distributed Scheduling of Virtual Machines Using Workload-Aware Consolidation Techniques," *Journal of Supercomputing*, vol. 61, no. 1, pp. 46-66, 2012.

[27] The Amazon Instance Types, https://aws.amazon.com/ec2/instance-types, Last Visited, 2017.

[28] The OpenStack Neat framework, http://openstack-neat.org, Last Visited, 2017.

[29] The OpenStack platform, http://openstack.org, Last Visited, 2017.

[30] Verma A., Ahuja P., and Neogi A., "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," *in Proceedings the 9th ACM/IFIP/USENIX International Conference on Middleware*, Leuven, pp. 243-264, 2008.

[31] VMware Inc., "VMware Distributed Power Management Concepts and Use," Information Guide, 2010.

[32] Wang X. and Wang Y., "Coordinating Power Control and Performance Management for Virtualized Server Clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 245-259, 2011.

[33] Weber W., Fan X., and Barroso L., "Powering the Data Center," United States Patent No. 8595515, 2013.

[34] Zheng W., Bianchini R., Janakiraman G., Santos J., and Turner Y., "JustRunIt: Experiment-Based Management of Virtualized Data Centers," *in Proceedings of the USENIX Annual Technical Conference*, San Diego, pp. 18-33, 2009.

[35] Zhu X., Young D., Watson B. J., Wang Z., Rolia J., Singhal S., McKee B., Hyser C., Gmach D., Gardner R., Christian T., and Cherkasova L., "1000 islands: Integrated Capacity and Workload Management for The Next Generation Data Center," *in Proceedings of the 5th International Conference on Autonomic Computing*, Chicago, pp. 172-181, 2008.

Loiy Alsbatin is a Ph.D. student in the department of Computer Engineering in Eastern Mediterranean University (EMU), North Cyprus. He received his B.S. degree in Computer Engineering in Mutah University, Jordan, in 2008, and his M.S. degree in Computer Engineering in Jordan University of Science and Technology (JUST), Jordan, in 2012. He is currently a faculty member at the Computer Science Department of Shaqra University, Saudi Arabia. His current research interests include distributed system, and cloud computing.

Gürcü Öz received her B.S, M.S. degrees from the Electrical and Electronic Engineering department and Ph.D. degree from the Computer Engineering Department of Eastern Mediterranean University, in Famagusta, North Cyprus. Currently, she is working as an Associate Professor in the Department of Computer Engineering of Eastern Mediterranean University. Her research interests include computer networks, wireless networks, distributed systems, cloud computing, system simulation, information security.

Ali Ulusoy was born in Eskisehir, Turkey, on June 3, 1974. He graduated from the double major program of the department of Electrical and Electronic Engineering (EEE) and department of Physics in Eastern Mediterranean University (EMU) in 1996. He received his M.S. and Ph.D. degrees in EEE in EMU in 1998 and 2004, respectively. He joined Information Technology department, EMU, in 2004. His current research interests include wireless communications, receiver design, channel estimation, fuzzy systems, wireless networks, cloud computing, millimeter wave communications and healthcare system development.