TARGO: Transition and Reallocation Based Green Optimization for Cloud VMs

Daren Fang, Xiaodong Liu
School of Computing
Edinburgh Napier University
Edinburgh, UK
{d.fang, x.liu}@napier.ac.uk

Lin Liu
School of Software
Tsinghua University
Beijing, China
linliu@tsinghua.edu.cn

Hongji Yang
Software Technology Research Lab
De Montfort University
Leicester, UK
hyang@dmu.ac.uk

Abstract* — Much research has been conducted focusing on improving resource utilization efficiency in data centers in the context of Green Cloud Computing (GCC). While virtualization enables better resource provision and utilization for various computational resources, different approaches are proposed based on virtual machine (VM) optimizations using either server or workload consolidation techniques. Nonetheless, these solutions can only be applied inside the Cloud. In fact, Infrastructure-as-a-Service (IaaS) users can hardly proactively achieve better VM resource utilization efficiency, as they typically have no control over any hypervisor or hardware in any Clouds. The issue gets more critical when workloads on VMs alter dramatically from time to time. This paper presents a novel approach namely Transition and Reallocation Based Green Optimization (TARGO) for such users. Through fully automated and intelligent VM optimization according to customizable optimization rules, TARGO guarantees that VMs or their successors being optimized will always run at their customizable green optimal states regardless how workloads vary. Experiments conducted on Amazon EC2 instances in the EU region show that, even under extreme random workloads, TARGO is still capable of selecting and retaining VM successors which run at an average CPU utilization of 50%-60%.

Keywords* - Green Cloud Computing; Server Consolidation; VM Migration; Green Optimization Rules; IaaS;

I. INTRODUCTION

In a Cloud Computing (CC) environment, resource virtualization organizes CC hardware components into independent resource blocks where workloads are handled within virtualized boundaries. In this context, much GCC research has been done addressing on virtualization and VM management optimizations. Current approaches to achieve better energy and (computational) resource utilization efficiency can be generally classified into two categories: 1) server/workload consolidation solutions involving VM scheduling [1][14][16], migration [15][18][26], and sizing/resizing management [9][22][24]; 2) hardware integrated solutions involving dynamic voltage and frequency scaling (DVFS) [25] and hardware cooling control mechanism [12]. As a matter of fact, these approaches can only be implemented on Cloud service provider (CSP) side (inside the Cloud). For IaaS users who have extremely limited privilege and control over the Cloud VM hypervisor or any physical hardware, they are not applicable. In this paper, we present the design and implementation of TARGO, a pure CC client-side solution which facilitates IaaS VM transition and workload reallocation according to customizable optimization rules. TARGO is developed to work seamlessly with IaaS CSPs by allowing intelligent and fully automated VM green optimizations.

As a transition and reallocation based solution, TARGO periodically monitors resource utilization of the VMs and extracts their dynamic workload patterns. Whenever it detects concrete resource utilization/workload changes for a certain VM, TARGO initiates an optimization by dynamically replacing such with a successor of an optimal VM size (so that the successor can handle the current workload efficiently). All successors created will appear to be the same as the original VMs as they are created based on the most current VM images. At the end of such optimizations, workloads are reallocated to the successors once they become fully operational, without any interruption or gaps except the switching moment. The whole optimization process is automatically operated and logged once optimization rule parameters are defined. In general, the proposed approach has the following contributions:

• The novel Cloud client-side VM optimization enables automated VM transitions where selected successors always run within their own specified green boundaries no matter how individual workload varies. The approach is well capable to work on top of other server/workload consolidation optimization solutions.

• The dynamic workload pattern extraction and recognition provide regulated views of the VM resource utilization states, which reveal actual workload trends and concrete changes.

• The concept of Performance Gap Ratios (PGR) between different VM sizes in IaaS Clouds is introduced. From the performance test data of EU Amazon EC2 instances, a series of PGRs are calculated and demonstrated.

• The approach allows IaaS users to always retain the smallest VMs (VM with fewest resources) depending on the actual real-time workload. In comparison with using a fixed large-enough VM to manage fluctuated workload, it saves significant amount of computational resource, and enables better resource allocation and distribution inside IaaS Cloud.

The rest of the paper is organized as follows. Section II discusses the related work. Section III outlines system architecture and components of TARGO. In section IV, TARGO optimization rules are further explained. Section V demonstrates the implementation of TARGO. A series of experiments and evaluations are illustrated in Section VI. Finally, Section VII concludes the paper.

II. Related Work

In [1], the authors propose to optimize VM scheduling over Cloud data center based on energy-aware resource provision and allocation algorithm, where Green Cloud Architecture involving "energy-aware power model" is demonstrated. Power Aware Meta Scheduler [16] is another attempt implemented from the perspective of VM scheduler. By predicting Cloud data center utilization, the scheduler manages lifecycle of the running VMs through dynamic power saving stage switches. Earlier work based on similar approaches can also be found in [2][8]. Nonetheless, they all share a single limitation: none of them clarifies the time and cost needed for VM migration or state changes, which is a critical factor affecting the frequency and effectiveness of such optimizations [7]. Additionally, they do not consider comprehensive VM resource utilization (including CPU, memory, network, and disk usages), which would also influence the overall green efficiency of the optimized physical machines (PMs). A more comprehensive work is [14], where the authors argue a scheduling policy which models and manages virtualized data centers to maximize their utilization efficiency by considering the following facets: power consumption, service level agreements (SLAs), virtualization overheads, heterogeneity management, etc. This provides a comparatively comprehensive view of the VM scheduling approach.

Dynamic resource allocation based on VM migration [26] attempts to minimize the overall number of running PMs in a server pool according to relevant load prediction, resources utilization distribution and "hot to cold" threshold algorithms. Basically, after the allocation and migration decision is made, VMs of different load are properly migrated between the PMs, where idle PMs can be resulted and then are turned off. Similar approaches are seen in [15][18]. Differences are that instead of workload predictions, monitor services and power meter (in [15]) and service analyzer and energy monitor (in [18]) are used to facilitate VM migration decisions. Another work is the resource allocation optimization system [10]. By employing an application prediction module which estimates SLA relevant resource utilization, the system performs VM migration based PM "mutation" actions. The overall efficiency is optimized due to the fact that VM resource utilizations are effectively managed across all PMs. Nevertheless, a possible drawback of all above approaches is that they fail to consider the effort and energy consumed while performing VM migration in such large scales, such as the overall time and total power usage of those large volumes of VM migrations [11].

Another perspective in server consolidation approaches is known as VM sizing/resizing, which is argued in [9][24] [22]. EnaCloud [9] comprises two key components: Global Controller which manages job scheduling and distribution tasks, and Resource pool where VM hypervisors perform monitor, sizing, and power management control. GARL [24], on the other hand, is equipped with VM Power Monitor and VM Resource Allocator which facilitate VM resizing and replacement according to VM performance monitor data. The authors of [22] utilize workload forecasting statistics as the basis to facilitate VM sizing and resizing consolidation tasks.

For software hardware combined optimizations, CloudScale [25], an automatic elastic resource scaling system (for multi-tenant Cloud infrastructure), consists of three key aspects: service level objective (SLO) based resources demand prediction, support of concurrent multi-tenant VM migration, and integration of DVFS scaling control. Another work [12] introduces DVFS along with active fan cooling control mechanism into VM scheduling tasks to achieve better resource utilization efficiency. iSIM framework [17] presents an alternative for VM performance optimization via virtual CPU core state management depending on the dynamics of the workload

From a different viewpoint, dynamic workload placement optimization is found as another solution. Discrete Particle Swarm Optimization (DPSO) algorithm [23] is proposed as a solution to allocate/distribute appropriate workload across VMs with different CPU cores. The authors of [13] advocate artificial swarm intelligence as well as Ant Colony Optimization (ACO) algorithm for a more energy efficient workload placement.

In contrast to the above work, TARGO focuses on improving resource utilization and consumption of VMs provisioned in the Cloud. Specifically for IaaS users who rarely have control over the Cloud hypervisor and hardware nor the workload, it actively replaces the non-green running VMs to those with better (more suitable) resources allocated. As a pure client-side VM-oriented solution, TARGO is able to work well on top of other approaches to achieve additional resource saving. Additionally, considering the significant differences between VMs due to their performance gaps as well as distinct types/patterns of workloads in real world scenarios, TARGO employs dynamic workload pattern extraction as well as customizable optimization parameters, so that accurate and effective optimizations can be facilitated regardless how VMs' performances and workloads vary.

Furthermore, for some IaaSs, there are some native solutions regarding resource monitor and utilization of VMs. Examples for Amazon EC2 are seen as CloudWatch, Auto Scaling and Elastic Load Balancing. CloudWatch [3] provides monitor functionalities for many services in AWS. It allows EC2 users to view instance (and other) utilization statistics according to customizable metrics, such as CPU usage, network or disk I/O information, etc. Additionally, with monitor alarms setup, it can automatically perform a series of simple actions according to users' configuration once an alarm is triggered, e.g., EC2 users can configure alarms to send them notification emails if selected VM instances' CPU utilization remains at 100% for a certain period. Auto Scaling [4] is designed to enable scaling optimization options for EC2 instances. According to users' configurations, it acts automatically to increase/decrease the number of running EC2 instances based on either periodic "healthy" checks of the selected current running instances at fixed schedules. Elastic Load Balancing [5] works to balance the network traffic load of EC2 instances, so that workloads are distributed evenly rather than concentrating on certain ones. Although the above three services seem to be very useful, in fact, they are not primarily designed for green effectiveness and efficiency requirements: I) At the moment, AWS alarm mechanism for EC2 only provides three available actions if an alarm is triggered: to send notification email(s), to shut down the instance(s), to terminate the instance(s). These options obviously do not switch any instance into a "green mode". In EC2 control panel, there is an option to change the instance size, but the instance must be stopped first to perform this resizing task. II) Auto Scaling can only launch new instances from the same AMI, whereas all new instances created have to be of the same instance size (of the original instance). III) Elastic Load Balancing simply distributes network traffics evenly, across different regions and availability zones.

In comparison with CloudWatch, TARGO acts dynamically to find appropriate VM successors according to the dynamics of the workload. By doing so continuously, it always retains instances in most green efficient sizes so that all VMs can run their own workload within own specified green boundaries. In contrast to Auto Scaling, TARGO launches successors in different machine sizes, in fact, the most green-effective sizes. Compared with adding/removing instances at a per instance basis and then to distribute the workload, TARGO saves VM resources as each optimization can be performed at a smaller scope. In many cases an appropriate larger/smaller sized instance would be able to manage the increased/decreased work at better green efficiency whilst several same-sized instances could be easily over-provisioned.

III. System Archetecture

A. Rule-based Intelligent Optimization

The architecture of TARGO is shown in Fig. 1. As a solution to inefficient VM resource utilization, TARGO behaves proactively to implement intelligent VM size transitions and workload reallocations, according to a series of optimization rule parameters. Generally, such optimization comprises the following steps: Whenever there is excessive high or low workload detected on a VM, which indicates that the instance cannot handle it green efficiently, a successor VM of larger or smaller type is created using its most current VM image. While the new better-sized VM is up and running, the original optimization rule parameters are dynamically adjusted to fit the successor VM as well. Finally, the new rule and the workload are transferred to the new VM where the original one is switched off. The whole optimizations process takes as less as few minutes. To the end users, there would be only millisecond's unavailable state for the moment of the switch, but the dynamically retained VMs are guaranteed to operate at their optimal green states.

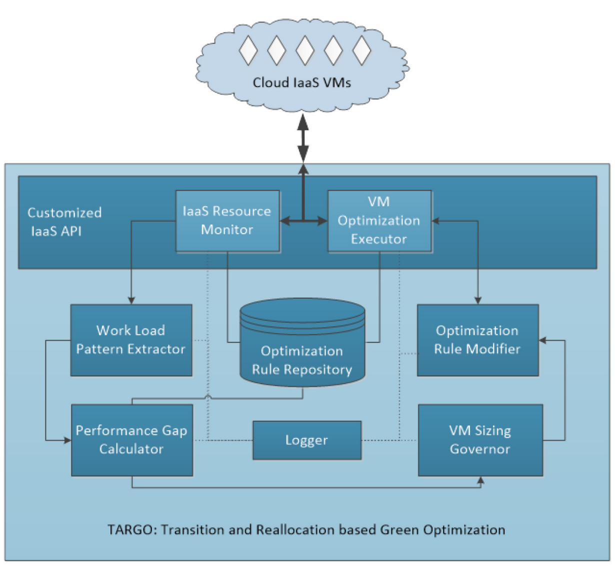

Cloud IaaS VMs
Customized IaaS API
IaaS Resource Monitor
VM Optimization Executor
Work Load Pattern Extractor
Optimization Rule Repository
Optimization Rule Modifier
Performance Gap Calculator
Logger
VM Sizing Governor
TARGO: Transition and Reallocation based Green Optimization

Figure 1. TARGO System Architecture

B. Workload Pattern Extractor

Through IaaS resource monitor interfaces provided by many CSPs, statistics of a series of VM resource metrics are available at certain costs, e.g., CPU usage, network, disk in/out, etc. These are used as raw data to formulate dynamic workload patterns. Here, a workload pattern can be understood as: for a specific repeating interval, the load of a VM seems to fit for certain regularity studies. However, due to the complexity of various real world VM applications, some may not come in fixed patterns. In addition, performances of VMs could vary depending on their images, type specifications, geographic locations, or even at different time slots. In fact, while different VMs are running their tasks, even for the same percentage increase on their own workloads, they could react differently. Therefore, load prediction algorithms and fixed load pattern modeling would hardly work. To effectively deal with the above issues, the dynamic workload pattern extractor is introduced to lively reflect the current status of the VM instances. Based on a customizable monitor frequency in a repeating period, a regulator is used to produce the most accurate utilization data.

$$RV^t = \frac{F}{P} * \sum_{n=1}^{P/F} MV_n^{tn} \qquad (1)$$

Equation (1) is used to compute the regulated monitor statistics so as to formulate workload patterns. F and P specify monitor frequency and period respectively. RV^t stands for regulated value at time t for certain F and P. MV_n^t indicates the n^{th} raw monitor data value recorded at each monitor time t_n. Basically, for each monitor interval, the average value of all values collected from the last monitor period is used as the regulated value. Generally speaking, the more stable certain workload is expected, the smaller P

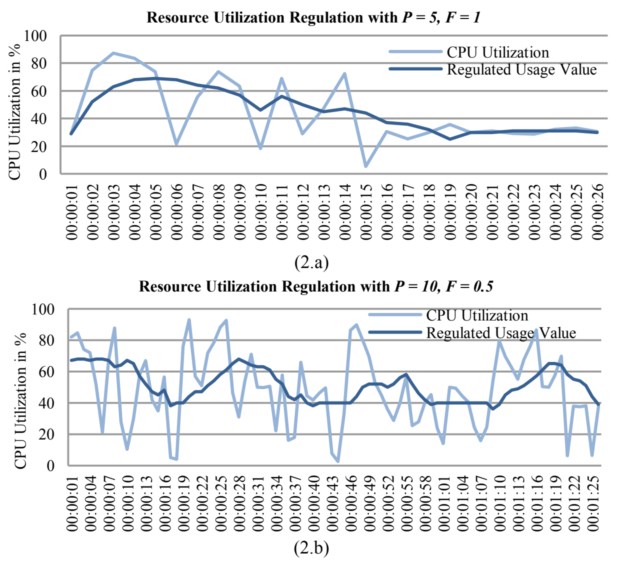


Figure 2. Workload Pattern Extraction

and larger *F* should be set to; the more dynamic the workload is anticipated, the larger *P* and smaller *F* should be given. Fig. 2 shows two examples of how raw CPU monitor data in percentage are regulated in Workload Pattern Extractor based on (1). It can be seen that the regulated utilization data appears to be concrete and is capable of showing the trend of the workload.

C. *Performance Gap Calculator*

Nowadays, most CSPs like Amazon EC2 and Rackspace Cloud Servers offer more than five size/type options in different platforms/OSs (operating system), which range from small to very large and are designed and provisioned to handle different tasks and workload. To facilitate green optimization for IaaS VM by making sizing transitions, it is essential to measure the approximate gaps of performances between the available VM sizes.

EC2 Instance Types:	**CPU: Super Pi (1M) Calculation Time**	**wPrime (32M) Calculation Time**	**Performance Gap Ratio:**
T1.Micro	*170.751 sec*	*478 sec*	***0.24***
M1.Small	*42.657 sec*	*158.021 sec*	***0.51***
M1.Medium	*21.148 sec*	*90.339 sec*	***0.57***
M1.Large	*18.174 sec*	*35.194 sec*	***0.69***
M1.Extra Large	*18.125 sec*	*21.909 sec*	***0.61***
M3.Extra Large	*12.724 sec*	*18.717 sec*	***0.71***
M3.Double Extra Large	*12.711 sec*	*9.367 sec*	

Table 1. EU Amazon EC2 Instance CPU Performances and PGRs

To find out how exactly Cloud VM instances of different sizes would respond to a same workload, a number of performance tests were implemented on Amazon EC2 instances (with plain Windows Server 2012 image in EU region) to examine their CPU frequency and multi-core scores. Although similar work evaluating KVM virtualization performance is done in [19], we implemented own comprehensive ones over Amazon EC2 Xen virtualized infrastructure due to later experiments needs. Super Pi (single core processing ability) [20] and wPrime (multi-core compute capability) [21] were used to test how quickly heavy compute tasks could be completed. Generally, the fewer the time needed, the better the virtual processor performance is. Table 1 demonstrates the (best) average performance test data of 20 concurrent tests executed at different time during the day (Data was correct at the time of the tests, although may vary a little at a different time). By comparing the result, the equivalency ratios were then "approximately" extracted.

The last column of Table 1 presents the ratios between different VM types of the EU EC2 instances. This is to be understood as: a workload resulting in 50% CPU usage on an m1.medium VM instance would be roughly equivalent to 100% of such in size of m1.small and 30% for an m1.large. Note that the ratios indicate general equivalency information, and may fluctuate under certain circumstances.

D. *VM Sizing Governor*

VM Sizing Governor generates transition calls when a VM reaches its thresholds in limits according to one's current workload and certain PGR. Currently TARGO only supports step-by-step transitions. Although this may slow the transition pace down in case of experiencing dramatic workload alteration, the overall effectiveness of the optimization is affected rarely.

E. *Optimization Rule Modifier*

Optimization Rule Modifier manages the consistency of rule customization based on PGRs for every VM successor returned. Currently it only changes the green up/down limits where necessary. The reason for rule modification is that after several times of optimization transitions, the original specified green boundary can become too wide or too narrow for a specific VM size of the current successor. If so, it would cause to either miss the optimal transition opportunity or start an optimization earlier than it should be. Therefore, without this proactive green limits adjustment for the new VM, TARGO would not run effectively. The modification process initiates immediately after a successor reaches its running state and successfully takes place of the original instance.

F. *Optimization Executor*

Optimization executor implements calls generated by VM Sizing Governor by sending request to IaaS providers. This involves retrieving complete VM specification information, creating the current VM image, getting state of the image created, creating a VM from the image as a successor, getting state of the successor, transferring bound information (e.g., IP address, customized names, tags, etc.) to the successor. In addition, to deal with the possible failures that may occur during the above process, Optimization Executor involves a series of error handling mechanisms which guarantee any optimization to be intelligent and consistent.

IV. Optimization Rules

A. *Optimization Rule Parameters*

An optimization rule has two parts. The first part involves ID, time (when a rule is created), and name of a rule; the second part is the body with following parameters: "Period" expresses the likely repeating period of how often one's workload would increase/decrease by a concrete level. "Frequency" indicates how often TARGO requests for raw Cloud monitor data for an instance. This also influences the frequency of workload pattern extraction, and so as the optimization frequency. "Up Limit" and "Down Limit" state a green boundary, only within which an instance is considered to be running at its optimal state. "Thresholds" indicates how many times that an instance can go below/over its down/up limits (after those times of violations, an instance is then regarded as necessary to be optimized). "Metric" specifies the name from available metric types such as CPU utilization, network in or out data, disk write or read operations, etc (this paper only focuses on the CPU metric). "Statistics" states how a selected metric is sampled: e.g., percentages of CPU utilization, bytes of network in/out data, times of disk read/write operations, etc. Metric and statistics options are set by and vary between different CSPs. Finally, "Counters" are up-to-date violation times of an instance has gone below/over its green limits. They are refreshed every time when there is a violation.

B. *Optimization Rule Modification*

1) *Rule Modification: To Broaden Out*

Due to the fact that IaaS users have only a few fixed VM size options, whenever a transition is initiated and the best successor is successfully returned, we must check how the workload would drive the CPU utilization on the successor, and make modifications if necessary. In fact, while upsizing/downsizing, the original down/up limit may become too large/small once certain optimization(s) occur(s). Either of those situations would cause TARGO to run into an infinite transition state.

2) *Rule Modification: To Narrow Down*

On the other hand, it is necessary to narrow down the gap of the green up/down limits, so that transitions can always take place at the optimal moments. Generally, as a VM is upsizing, one's overall performance is getting steadier. As a result, neither a narrow boundary set on a small sized VM nor a wide boundary set on a large sized VM would be appropriate. An example is that an initial green boundary of 20%-80% is reasonably settled on a t1.micro VM, and after a number of transitions the boundary is transferred into a successor in size of m1.large. It then would become too wide for the m1.large instance as there will not be any optimization as long as the majority of its CPU utilization stays between 10%-90%.

3) *Modification Equations*

If Counter >= T_U:

$$NL_U = OL_U; \tag{2.1}$$

$$NL_D = \begin{cases} NL_U * Ratio_{M(O-N)};\ (if\ OL_D > NL_U * Ratio_{M(O-N)}) \\ NL_U * Ratio_{M(O-N)}^2;\ (if\ OL_D < NL_U * Ratio_{M(O-N)}^2) \\ OL_D;\ (else) \end{cases} \tag{2.2}$$

If Counter >= T_D:

$$NL_U = \begin{cases} \frac{NL_U}{Ratio_{M(O-N)}};\ \left(if\ OL_U < \frac{NL_D}{Ratio_{M(O-N)}}\right) \\ \frac{NL_U}{Ratio_{M(O-N)}^2};\ \left(if\ OL_U > \frac{NL_D}{Ratio_{M(O-N)}^2}\right) \\ OL_U;\ (else) \end{cases} \tag{3.1}$$

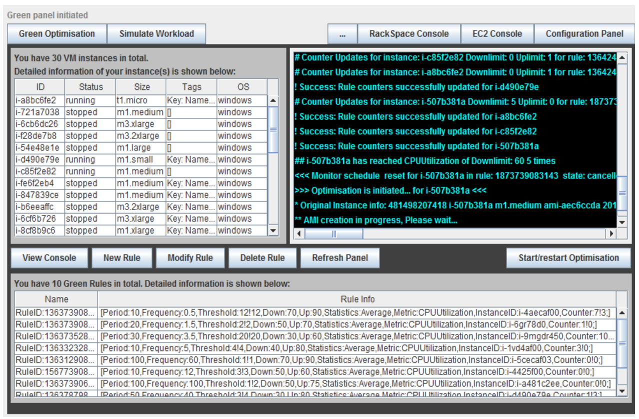
$$NL_D = OL_D \tag{3.2}$$

Where $NL_U <= 90$ and $NL_D >= 10$. U means Up and D means Down. OL stands for the original limit, whereas NL stands for the new limit. T is the threshold value. $Ratio_{M(O-N)}$ means the respected PGR of the metric (CPU) between the original instance size and the new size.

Equations (2.1) to (3.2) illustrate how the limits should be customized based on the current instance size and the relevant PGR. Basically, I) for upsizing, the up limit remains unchanged, whereas the down limit should be a value between multiplying the current limit by the first and second power of the respected PGR from the original VM size to the new size; II) for downsizing, the down limit stays unaltered; whereas the up limits should be between dividing the current limit by the first and second power of the respected PGR. In this way, no matter how a workload scales up or down, it is guaranteed that after the transition there is a properly updated green boundary for the successor where the workload will fall within the limits (as long as it is not suddenly changed).

V. Implementation

Shown in Fig. 3, the TARGO prototype provides a full optimization console as well as additional IaaS management interfaces. Initially, the system asks a user to enter one's IaaS API access security credential. In the optimization panel one can create new optimization rules by selecting target instances and entering mandatory parameters. Later modification or deletion can be made. By clicking the "start/restart Optimization" button, the optimization processes are initiated. Every VM in optimization rules has its private controller and runs in its individual optimization cycle. The whole system runs fully automatically and does not need any supervision. Whenever an optimization occurs,



You have 30 VM instances in total.
Detailed information of your instance(s) is shown below:

ID	Status	Size	Tags	OS
i-a8bc6fe2	running	t1.micro	Key: Name...	windows
i-721a7038	stopped	m1.medium	[]	windows
i-6cb6dc26	stopped	m3.xlarge	[]	windows
i-f28de7b8	stopped	m3.2xlarge	[]	windows
i-54e48e1e	stopped	m1.large	[]	windows
i-d490e79e	running	m1.small	Key: Name...	windows
i-c85f2e82	running	m1.medium	[]	windows
i-fe6f2eb4	stopped	m1.medium	Key: Name...	windows
i-847839ce	stopped	m1.medium	Key: Name...	windows
i-b6eeaffc	stopped	m3.2xlarge	Key: Name...	windows
i-6cf6b726	stopped	m3.xlarge	Key: Name...	windows
i-8cf8b9c6	stopped	m1.xlarge	Key: Name...	windows

```
# Counter Updates for instance: i-c85f2e82 Downlimit: 0 Uplimit: 1 for rule: 136424
# Counter Updates for instance: i-a8bc6fe2 Downlimit: 0 Uplimit: 1 for rule: 136424
! Success: Rule counters successfully updated for i-d490e79e
# Counter Updates for instance: i-507b381a Downlimit: 5 Uplimit: 0 for rule: 187373
! Success: Rule counters successfully updated for i-a8bc6fe2
! Success: Rule counters successfully updated for i-c85f2e82
! Success: Rule counters successfully updated for i-507b381a
## i-507b381a has reached CPUUtilization of Downlimit: 60 5 times
<<< Monitor schedule  reset for i-507b381a in rule: 1873739083143  state: cancell
>>> Optimisation is initiated... for i-507b381a <<<
* Original Instance info: 481498207418 i-507b381a m1.medium ami-aec6ccda 201
** AMI creation in progress, Please wait...
```

You have 10 Green Rules in total. Detailed information is shown below:

Name	Rule Info
RuleID:136373908...	[Period:10,Frequency:0.5,Threshold:12!12,Down:70,Up:90,Statistics:Average,Metric:CPUUtilization,InstanceID:i-4aecaf00,Counter:7!3;]
RuleID:136373908...	[Period:20,Frequency:1.5,Threshold:2!2,Down:50,Up:70,Statistics:Average,Metric:CPUUtilization,InstanceID:i-6gr78d0,Counter:1!0;]
RuleID:136373528...	[Period:30,Frequency:3.5,Threshold:20!20,Down:30,Up:60,Statistics:Average,Metric:CPUUtilization,InstanceID:i-9mgdr450,Counter:10...
RuleID:136332328...	[Period:10,Frequency:5,Threshold:4!4,Down:40,Up:80,Statistics:Average,Metric:CPUUtilization,InstanceID:i-1vd4af00,Counter:3!0;]
RuleID:136312908...	[Period:100,Frequency:60,Threshold:1!1,Down:70,Up:90,Statistics:Average,Metric:CPUUtilization,InstanceID:i-5cecaf03,Counter:0!0;]
RuleID:156773908...	[Period:10,Frequency:12,Threshold:3!3,Down:50,Up:60,Statistics:Average,Metric:CPUUtilization,InstanceID:i-4425f00,Counter:0!0;]
RuleID:136373906...	[Period:100,Frequency:100,Threshold:1!2,Down:50,Up:75,Statistics:Average,Metric:CPUUtilization,InstanceID:i-a481c2ee,Counter:0!0;]
RuleID:136378798...	[Period:50,Frequency:40,Threshold:3!4,Down:30,Up:80,Statistics:Average,Metric:CPUUtilization,InstanceID:i-d490e79e,Counter:1!3;]

Figure 3. Screenshot of TARGO Prototype

the bound information (IP address, tag, name, etc.) is not transferred to the successor until it reaches the full operational state. This guarantees minimal side effects of the optimization.

As a matter of fact, the life-cycles of VMs in Clouds are far different from those in a private infrastructure environment. The time needed for VM creation, booting-up and shutting-down, image creation, etc. are highly variable. Not to say that these processes even sometimes fail. To deal with these problems, TARGO is built with a series of error handling components and is therefore capable of intelligently solving various unexpected incidents. This ensures service consistency of the VMs being optimized. In addition, every action TARGO implements is logged, which helps users understand all background processes of optimizations. Fig. 4 demonstrates an example log saved while an optimization occurred. Used Amazon EC2 instances shown in the example, on reaching the threshold specified in the rule, TARGO retrieved all information of the instance and then selected an appropriate successor size at first. The AMI creation request was then sent. As the AMI became available, successor creation was started. When the successor became up and fully available, the follow-up transfer was performed, involving tag and IP re-association.

```
…
Sat Mar 23 14:17:30 GMT 2013 # For i-d490e79e of CPUUtilization RV: 73 Sat Mar 23
14:17:30 GMT 2013
Sat Mar 23 14:17:30 GMT 2013 # For CPUUtilization Due to: 73 >= 70, Uplimit Counter
updated in rule list
Sat Mar 23 14:17:30 GMT 2013 # Counter Updates for instance: i-d490e79e Downlimit: 3
Uplimit: 4 for rule: 1363739083143
Sat Mar 23 14:17:30 GMT 2013 ! Success: Rule counters successfully updated for i-d490e79e
Sat Mar 23 14:17:30 GMT 2013 ## i-d490e79e reached CPUUtilization Uplimit: 70 4 limits
Sat Mar 23 14:17:31 GMT 2013 <<< Monitor schedule    reset for i-d490e79e in rule:
1363739083143     state: cancelled >>>
Sat Mar 23 14:17:31 GMT 2013 >>> Optimisation is initiated... for i-d490e79e <<<
Sat Mar 23 14:17:33 GMT 2013 * Original Instance info: 481498207418 i-d490e79e m1.small
ami-800c04f4 2013CCS quick-start-3 176.34.184.176 Name iis
Sat Mar 23 14:17:33 GMT 2013 **Instance found, Start creation process!**
Sat Mar 23 14:17:34 GMT 2013 ** AMI creation in progress, Please wait...
Sat Mar 23 14:17:35 GMT 2013 *** SUCCESS! AMI of original instance: ami-aec6ccda is
ready for use. Start new instance creation...
Sat Mar 23 14:17:36 GMT 2013 ***Succesor of i-d490e79e is i-507b381a m1.medium
ami-aec6ccda 2013CCS quick-start-3
Sat Mar 23 14:17:36 GMT 2013 * Awaiting new instance: i-507b381a to reach running state
Sat Mar 23 14:17:56 GMT 2013 * Awaiting new instance: i-507b381a to reach running state
Sat Mar 23 14:19:09 GMT 2013 * New instance: i-507b381a is fully ready for use...
Sat Mar 23 14:19:09 GMT 2013 ** Tags Reassociation of Name iis is successfully completed
between i-d490e79e i-507b381a
Sat Mar 23 14:19:10 GMT 2013 *** IP Reassociation on 176.34.184.176 is successfully
completed between i-d490e79e i-507b381a
Sat Mar 23 14:19:10 GMT 2013 ** Switch completed, original instance: i-d490e79e will be
stopped after certain dalay...
Sat Mar 23 14:19:14 GMT 2013 ** New instance: i-507b381a is successfully returned
Sat Mar 23 14:19:15 GMT 2013 %% Rule modified for i-507b381a with EQ: 0.51
Sat Mar 23 14:19:15 GMT 2013 %% New rule prepared for the instance:
[Period:4,Frequency:0.5,Threshold:4!4,Down:36,Up:70,Statistics:Average,Metric:CPUUtilizatio
n,InstanceID:i-507b381a,Counter:0!0;]
Sat Mar 23 14:19:15 GMT 2013 !! Success: Rulelist updated with new rule applied. New
Monitor schedule will be started shortly for successor.
Sat Mar 23 14:19:15 GMT 2013 << New Monitor schedule created: i-507b381a for ruleID:    >>
Sat Mar 23 14:19:15 GMT 2013 <<< Succesor Monitor schedule for: i-507b381a started >>>
Sat Mar 23 14:19:15 GMT 2013 >>> Optimisation is complete... i-d490e79e is replaced with
new instance: i-507b381a <<<
Sat Mar 23 14:19:16 GMT 2013 # Awaiting monitor data... for instance: i-507b381a
Sat Mar 23 14:19:46 GMT 2013 # For i-507b381a of CPUUtilization {Timestamp: Sat Mar 23
14:20:10 GMT 2013 *** Original instance: i-d490e79e stopped...
Sat Mar 23 14:20:16 GMT 2013, Average: 77.5, Unit: Percent, }
Sat Mar 23 14:20:46 GMT 2013 # For i-507b381a of CPUUtilization RV: 42 Sat Mar 23
14:24:16 GMT 2013
Sat Mar 23 14:21:16 GMT 2013 # Updates: Monitor value is within limits for i-507b381a in
rule:
…
```

Figure 4. System Logs of TARGO

VI. Experiments & Evaluation

A series of experiments have been conducted over Amazon EC2 instances of different VM sizes in the EU region. To fully test the effectiveness and scalability of TARGO, the tests involved a series of CPU-oriented workloads with different dynamics of distinct patterns. The test workloads were generated by Apache Jmeter [6] installed on a number of client machines, which sent quantitative server requests in a controlled manner. On the VM server side, for every request that Jmeter sent out there was a calculation task executed. Additionally, for available VM sizes, as IaaS providers typically offer choices of both CPU cores and processor speed, we specifically adjusted the calculation task so that it is both CPU clock speed and multi-core sensitive.

In addition, in order to show the effectiveness of TARGO, we demonstrate statistics collected from a large range of instances available in AWS from t1.micro to m3.2xlarge. All (initial) instances (successor instances were originated from the latest images of last VM automatically) used in the experiments were created from Amazon's official "Windows 2008 Server SP2 with IIS" AMI.

A. Optimization under Stead Workload

Fig. 5 illustrates an example of using TARGO while experiencing a fairly steady workload. The round dots describe the volume of experiment workload allocated to a fixed IP addresses (yet it actually pointed to a series of VMs). The test started with an initial workload of one unit, and after continuous steady increase for every ten minutes passed, it finally ended with 15000% of the initial value. The IP address used in the experiment was firstly assigned to an m1.small EC2 VM instance with the test AMI image. During the test, it was re-associated again and again on each successor, until finally on the final one in size of m3.2xlarge. The initial rule for the original instance in m1.small was specified as "Period: 5, Frequency: 0.5, Threshold: 4/5, Down: 50, Up: 70, Statistics: Average, Metric: CPUUtilization". By the end of the experiments, the final

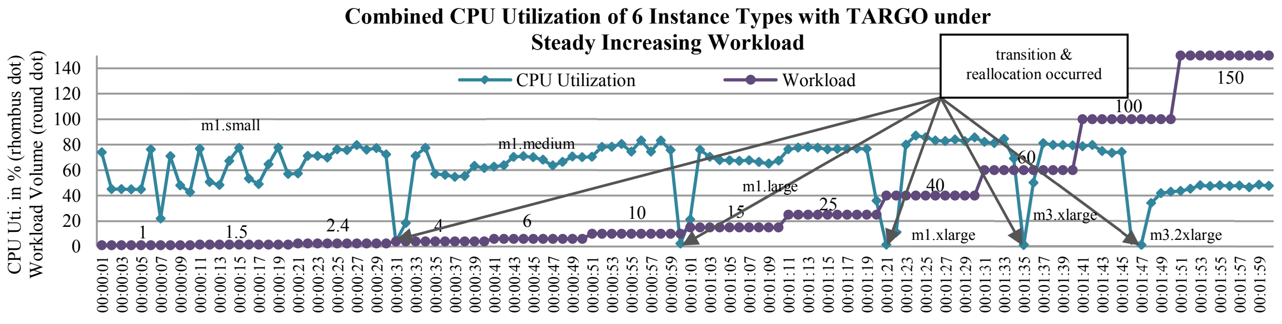


Figure 5. Optimization under Steady Increasing Workload

modified rule for the size of m3.2xlarge was "Period: 5, Frequency: 0.5, Threshold: 4/5, Down: 36, Up: 70, Statistics: Average, Metric: CPUUtilization".

The rhombus dots in Fig. 5 indicate the combined CPU utilization values monitored at a one minute interval for all instances involved during the optimization transitions. This indicates that each of those instances has participated in handling certain workloads during its running cycle. For example, the initial instance with size of m1.small was stopped at the time 00:00:31, whereas from then the workload started to run on the successor instance with size of m1.medium. At 00:01:00, the successor in m1.medium was replaced with a new successor in m1.large… until the final successor in m3.2xlarge. This proves that TARGO has successfully managed to: 1) find better instance types according to the quantity of real-time workload; 2) dynamically request and create successors in appropriate sizes by using the same AMI; 3) complete the transition and reallocation process without service interruption by transfer whatever is needed.

In this experiment, TARGO implemented five optimizations, considering how green efficient those instances were serving their duties, notably in Fig. 5, the CPU utilization values are generally between 40% and 80%. This suggests that each VM successor was working at a comparative green level while contributing own effort. Nonetheless, notably the 40%-80% contradicts the initial boundary of 50%-70%. This is due to: 1) TARGO's threshold counting algorithm, which would only act after a VM reaching the limits; 2) reallocation delay caused while waiting a successor to boot up (current instance continues serving until its successor becomes fully operational, even if it is out of the green boundary). However, they do not affect the overall effectiveness of the green optimization, as the it shows TARGO is capable of keeping a VM instance at a green optimal machine type (with average of 60% CPU usage overall) based on the dynamics occurred in the workload. In a word, TARGO works faultlessly for instances under a steady workload, despite the critical increase in the workload.

B. *Optimization under Workload With Small Dynamics*

Fig. 6 describes a comparative example of with/without TARGO while experiencing a slightly more "complicate" workload. The controlled experiment workload demonstrated in Fig. (6.a) was firstly allocated separately on m1.large, m1.xlarge, and m3.xlarge three instances of the same image without involving TARGO; and then the test was rerun on an m1.large instance with TARGO using the rule: "Period: 5, Frequency: 1, Threshold: 5/5, Down: 40, Up: 80, Statistics: Average, Metric: CPUUtilization".

In Fig. (6.a) , it can be seen that the workload lasted for 90 minutes (from the first request sent until the final reply received), whilst there were dramatic increase and decrease in the first thirty minutes. The increase over the first twenty minutes was almost four times of the initial units. From then the decrease was fairly gradual. Notably that the increase and the decrease were at a per minute basis, so the dynamics was far more frequent than the one in Fig. 5.

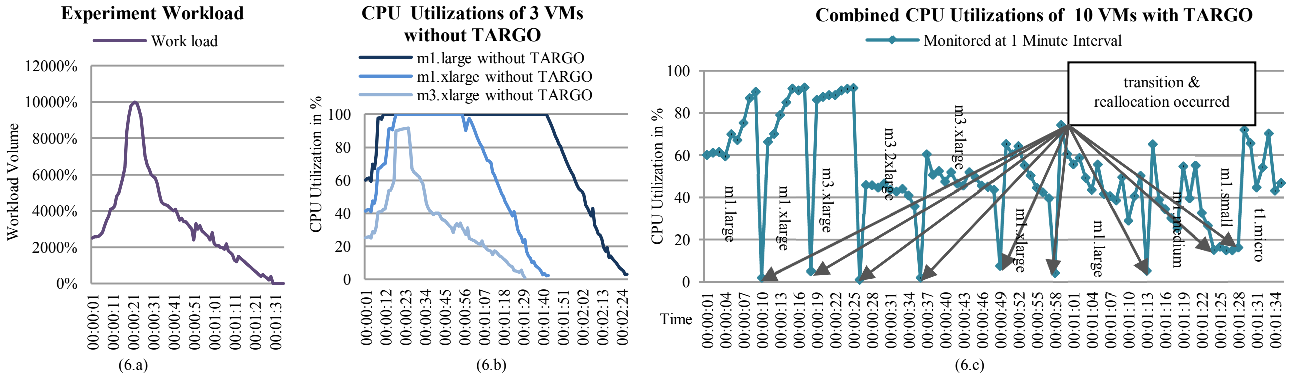


Figure 6. Optimization under More Dynamic Workload

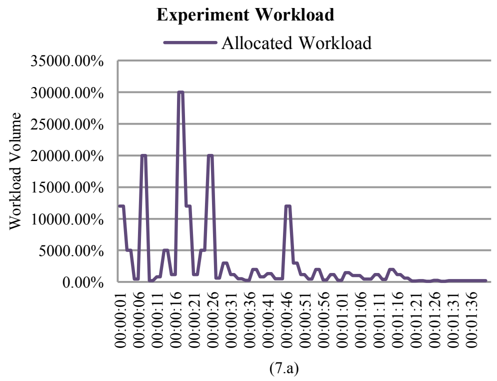
Seen in Fig. (6.b), although with the same test workload, only instance with the size of m3.xlarge managed to finish on time; the other two experienced certain delays due to their poorer performances. I) In size of m1.large, the VM instance took almost another hour to finish the whole test. From ten minutes after the test started, its CPU utilization remained at 100% until some decrease happened at one hour and forty minutes of time. This period was obviously not considered to be green friendly, as a constant full CPU load generally means the processor cannot handle the workload. II) With resource provision of m1.xlarge capacity, the VM also needed extra twenty minutes to complete the test. The constant CPU utilization of 100% for about half an hour indicates that the workload was still too heavy for this specification. Moreover, from thirty minutes before the VM finished the test, its CPU utilization dropped and remained below 50%, whereas for the last twenty minutes it did not even reached 20%. This implies that the workload during this period was too low for this instance size. III) The instance in m3.xlarge managed to complete the work in time. The resource capacity appeared to be acceptable for the workload. Yet, except for a small amount of time the CPU usage had stayed above 80%, the rest had remained under 40% for most of the test. Compared with m1.xlarge, the VM instance is over provisioned for the workload. This seriously failed to efficiently utilize the provisioned VM resources.

With TARGO, Fig. (6.c) illustrates the combined CPU usage data of all successors during the optimization transition processes. For the dynamics of the workload, the original instance in m1.large transited up and down nine times until it finally stayed at t1.micro when the workload stopped. The combine CPU utilization was generally between 30%-90%, despite the 40%-80% initially specified. This indicates that the original boundary was a little too wide for an m1.large, and after several times of rule modification, the final boundary turned into 20%-90% on the t1.micro. Still, the experiment result implies that TARGO is able to retain appropriate instances types whiling facing comparatively complicate workload which involves fast increase and decrease in a short period of time.

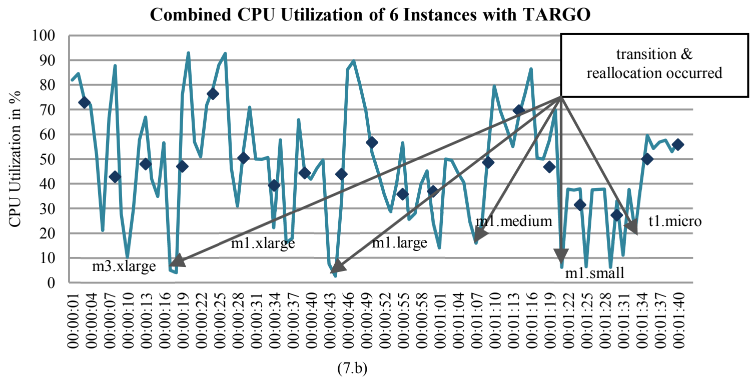
C. *Optimization under Extreme Random Workload*

Another experiment showing how TARGO would act under workload with extreme random patterns is demonstrated below in Fig. 7. Fig (7.a) illustrates the test workload, whilst Fig (7.b) shows the combined CPU utilization of the 6 VMs incurred during the transitions, with initial rule “Period: 10, Frequency: 0.5, Threshold: 4/4, Down: 50, Up: 70, Statistics: Average, Metric: CPUUtilization”. From Fig. (7.a) it can be seen that the workload used had dramatic increase and decrease in every two minutes, whereas the actual span of the unit difference was extremely large at 30000%. The only obvious change in the workload may be that it was decreasing in general considering the entire running time. However, for the first half an hour, there was not any clear trend. Most importantly, within some two minutes during the experiment, the actual increase/decrease was over 100 times. Clearly, no VM can run this workload with a definite steady CPU Utilization. Here, using a large enough (well-resourced) VM would be the only way to properly handle the peaks of such workload. However, this would also mean wasting considerable amount of resource due to over-provision for the rest of the load, which would fail to meet any green efficiency requirement.

On the other hand, even under such extreme dynamic workload, TARGO managed to find the best possible instance sizes to complete the tasks by maintaining their combined CPU utilization within a relatively green range. The initial instance was setup in size of m3.xlarge, whereas it was finally ended with a successor in t1.micro. Seen in Fig. (7.b), their combined CPU utilization data monitored at a per minute basis fluctuated seriously, from which it is hard to tell any clear judgment. Yet at a slightly larger monitor interval, the data then appear to be fall into a relatively small boundary. The square dots in Fig. (7.b) demonstrate the CPU utilization values monitored at every two minutes on each instances. Generally, almost all of the dots remain within the range of 35%-75%. This illustrates that even for a VM which works under extreme random workload that changes very frequently, TARGO is still capable of retaining successors in appropriate VM sizes according to the parameters entered in the optimization rule.



(7.a)



(7.b)

Figure 7. Optimization under Extreme Random Workload

VII. Conclusions & Future Work

While numerous research proposes server and workload consolidation as well as hardware integrated control mechanism solutions to achieve GCC, this paper argues another approach known as TARGO – the optimization specifically for IaaS VM users who usually have no control over any Cloud hardware or the hypervisor. We have demonstrated how TARGO could actively facilitate automated VM transition and workload reallocation optimizations on discovering waste or overuse of VM resources. Experiments have shown that TARGO could always retain suitable successors that would work within green boundaries regardless how workloads vary. The future work will address on: I) extended optimization scenarios based on additional metric types such as network, disk utilization; II) advanced optimization using third party VM load balancers on multiple VMs; III) multiple IaaS CSPs support.

Acknowledgment

The work in this paper has been sponsored by the Lawrence Ho Research Fund (LH-Napier2012).

References

[1] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for Cloud Computing”, *Future Generation Computer Systems*, vol. 28, issue 5, pp. 755-768, 2012

[2] A. Beloglazov and R. Buyya, “Energy Efficient Resource Management in Virtualized Cloud Data Centers”, *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp.826-831, 2010

[3] Amazon Web Services, Inc., “Amazon CloudWatch Developer Guide”, 2010, [Online], Available: http://awsdocs.s3.amazonaws.com/AmazonCloudWatch/latest/acw-dg.pdf, [Accessed: Feb. 21, 2013]

[4] Amazon Web Services, Inc., “Auto Scaling Developer Guide”, 2011, [Online], Available: http://awsdocs.s3.amazonaws.com/AutoScaling/latest/as-dg.pdf, [Accessed: Feb. 24, 2013]

[5] Amazon Web Services, Inc., “Elastic Load Balancing Developer Guide”, 2012, [Online], Available: http://awsdocs.s3.amazonaws.com/ElasticLoadBalancing/latest/elb-dg.pdf, [Accessed: Feb. 22, 2013]

[6] Apache Software Foundation, “Apache JMeter”, 2013, [Online], Available: http://jmeter.apache.org/, [Accessed: Mar. 15, 2013]

[7] A. Strunk, “Costs of Virtual Machine Live Migration: A Survey”, *IEEE Eighth World Congress on Services*, pp.323-329, 2012

[8] A. Younge, G. Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, “Efficient resource management for Cloud Computing environments”, *International Green Computing Conference*, pp.357-364, 2010

[9] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, “EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments”, *IEEE International Conference on Cloud Computing*, pp. 17-24, 2009

[10] C. Huang, C. Guan, H. Chen, Y. Wang, S. Chang, C. Li, and C. Weng, “An adaptive resource management scheme in Cloud Computing”, *Engineering Applications of Artificial Intelligence*, vol. 26, issue 1, pp. 382-389, 2013

[11] D. Aikema, A. Mirtchovski, C. Kiddle, and R. Simmonds, “Green Cloud VM Migration: Power Use Analysis”, *International Green Computing Conference (IGCC)*, pp. 1-6, 2012

[12] D. Lago, E. Madeira., and L. Bittencourt, “Power-aware virtual machine scheduling on Cloud using active cooling control and DVFS”, *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, vol. 2, 2011

[13] E. Feller, L. Rilling, and C. Morin, "Energy-Aware Ant Colony Based Workload Placement in Clouds", *Grid Computing (GRID), 12th IEEE/ACM International Conference,* pp.26,33, 21-23, 2011

[14] L. Goiri, J. Berral, J. Fitó, F. Julià, R. Nou, J. Guitart, R. Gavaldà, J. Torres, “Energy-efficient and multifaceted resource management for profit-driven virtualized data centers”, *Future Generation Computer Systems*, vol. 28 issue 5, pp.718-731, 2012

[15] L. Liu, H.Wang, X. Liu, X. Jin, W. H, Q. Wang, and Y.Chen, “GreenCloud: a new archetecture for green data center”, *Proceedings of the 6th International Conference Industry Session on Automatic Computing and Communication Industry Session*, pp. 29-38, 2009

[16] R. Jeyarani, N. Nagaveni, S. Sadasivam and V. Rajarathinam, “Power Aware Meta Scheduler for Adaptive VM Provisioning in IaaS Cloud”, *International Journal of Cloud Applications and Computing*, vol.1, issue 3, pp.36-51, 2011

[17] R. Jeyarani, N. Nagaveni, S. Srinivasan, and C. Ishwarya, “ISim: A Novel Power Aware Discrete Event Simulation Framework for Dynamic Workload Consolidation and Scheduling in Infrastructure Clouds”, Advances in Intelligent Systems and Computing, vol. 177, pp. 375-384, 2013

[18] R. Karthikeyan and P.C Hitra, “Novel Heuristics Energy Efficiency Approach for Cloud Data Center”, *IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, pp. 202 – 207, 2012

[19] Q. Chen, P. Grosso, K. Veldt, C. Laat, R. Hofman, and H.Bal, “Profiling energy consumption of VMs for green Cloud Computing”, *Ninth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pp. 768-775, 2011

[20] wPrime Systems., “Super Pi Single-Thread Benchmark”, 2012, [Online], Available: http://www.superpi.net/, [Accessed: Jan. 06, 2013]

[21] wPrime Systems., “wPrime Multi-Threaded Benchmark”, 2012, [Online], Available: http://www.wprime.net/, [Accessed: Jan. 06, 2013]

[22] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, “Efficient Resource Provisioning in Compute Clouds via VM Multiplexing”, *ICAC '10 Proceedings of the 7th international conference on Autonomic computing*, pp.11-20, 2010

[23] Y. Chen and S. Tsai, “Optimal Provisioning of Resource in a Cloud Service”, *IJCSI International Journal of Computer Science*, vol. 7, issue 6, 2010

[24] Y. Guo and X. Zhou, “Coordinated VM Resizing and Server Tuning: Throughput, Power Efficiency and Scalability”, *20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp.289-297, 2012

[25] Z. Shen, S.Subbiah, X.Gu, and J. Wilkes, “CloudScale: elastic resource scaling for multi-tenant Cloud systems”, *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Vol. 5, 2011

[26] Z. Xiao, W. Song, and Q. Chen, “Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment”, *IEEE Transactions on Parallel and Distributed Systems*, issue. 99, 2012