A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers

Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya

Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: sarehf@student.unimelb.edu.au, {amir.vahid, rnc, rbuyya}@unimelb.edu.au

Abstract—**One of the major challenges that cloud providers face is minimizing power consumption of their data centers. To this point, majority of current research focuses on energy efficient management of resources in the Infrastructure as a Service model and through virtual machine consolidation. However, containers are increasingly gaining popularity and going to be major deployment model in cloud environment and specifically in Platform as a Service. This paper focuses on improving the energy efficiency of servers for this new deployment model by proposing a framework that consolidates containers on virtual machines. We first formally present the container consolidation problem and then we compare a number of algorithms and evaluate their performance against metrics such as energy consumption, Service Level Agreement violations, average container migrations rate, and average number of created virtual machines. Our proposed framework and algorithms can be utilized in a private cloud to minimize energy consumption, or alternatively in a public cloud to minimize the total number of hours the virtual machines leased.**

I. Introduction

The numerous advantages of cloud computing environments, including cost effectiveness, on-demand scalability, and ease of management, encourage service providers to adopt them and offer solutions via cloud models. It in turn encourages platform providers to increase the underlying capacity of their data centers to accommodate the increasing demand of new customers. One of the main drawbacks of the growth in capacity of cloud data centers is the need for more energy to power these large-scale infrastructures. With ever increasing popularity of cloud computing, data centers energy consumption is anticipated to double [1].

In addition to traditional cloud services—Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)—recently a new type of service, called Containers as a Service (CaaS), has been introduced by Google[1] and Amazon Web Services. Containers can be considered a new revolution in the cloud era since containers are lightweight, easier to configure and manage, and can decrease the start-up time considerably. Docker[2] is a good example of a container management system.

CaaS lies between IaaS and PaaS: while IaaS provides virtualized compute resources and PaaS provides application specific runtime services, CaaS is the missing layer that glues these two layers together. As illustrated in Figure 1, CaaS services are usually provided on top of IaaS' virtual machines (VMs). A recent study [2] show that VM-Container configurations obtain close to, or even better performance, than native Docker (container) deployments.

[1]Google CaaS: https://cloud.google.com/container-engine/
[2]Docker: https://www.docker.com/

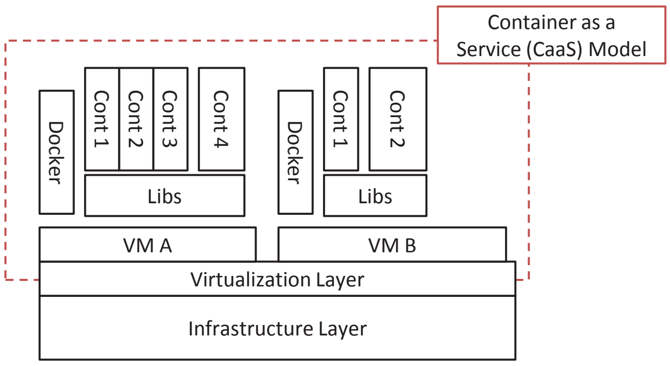


Fig. 1: Container as a Service (CaaS) model.

Although CaaS is increasingly gaining popularity and going to be one of the major cloud service models, improving energy efficiency in CaaS data centers has not yet been investigated deeply. Therefore, in this paper, we model the CaaS environment and the associated power optimization problem. Servers, cooling systems, and network elements are the three main power consumers in a data center, among which servers with 40-50% energy consumption has the biggest share [1]. Therefore, in this paper we propose a framework in CaaS context that decreases the number of running servers through container migration algorithms.

Like any consolidation solution, our framework should be able to tackle the consolidation problem in three stages. Firstly, it should identify the situations in which container migration should be triggered. Secondly, it should select a number of containers to migrate in order to resolve the situation. Finally, it should find migration destinations (host/VM) for the selected containers. In the first stage of this framework, container migration is initiated if a host is identified as over-loaded or under-loaded considering a specific criteria. Two algorithms with static under-load (UL) and over-load (OL) thresholds are utilized for initiating the migrations. For the second stage, "MCor" and "MU" algorithms are considered. "MCor" selects the containers having the most correlated workloads with the server's hosting the containers while "MU" selects the ones with the most CPU utilization. Finally for the third stage, three bin packing host selection algorithms, including First-Fit (FFHS), Random (RHS), and Least Full (LFHS), are applied. Additionally, a threshold-base correlation aware host selection algorithm named "CorHS" is also introduced. Later, the First-Fit algorithm is applied to select the destination VM that its capacity matches a predefined percentile of CPU workload of the container.

The framework and its algorithms are further evaluated through simulation. The widely used cloud simulation toolkit CloudSim [3] is extended to model a CaaS provider and implement the proposed framework. Real world workload traces from PlanetLab [4] is used as the containers' CPU workload. The experimental result shows that the "CorHS" policy that uses workload correlation analysis for destination host selection reduces the energy consumption in most of the scenarios.

The rest of the paper is organized as follows. Section II presents system objective and problem formulation . In Section III, the system architecture and its components are briefly discussed. Later in Section IV, the algorithms are presented. Section V discusses the testbed and the experiment results, while Section VI presents the related work. Finally Section VII discusses the conclusion of the work and possible future directions.

II. System Objective and Problem Formulation

In this section, we briefly discuss the objective of our proposed system, which is minimizing the data center overall energy consumption while meeting the Service Level Agreement (SLA). Firstly, we discuss the power model utilized for estimation of the data center energy consumption and the SLA metric used for comparison of consolidation algorithms. Symbols used in this section are defined in Table I.

A. Data Center Power Model

The power consumption of the data center at time t is calculated as follows:

$$P_{dc}(t) = \sum_{i=1}^{N_S} P_i(t) \qquad (1)$$

For estimation of power consumption of servers, we consider the power utilization of the CPU because this is the component that presents the largest variance in power consumption in regards to its utilization rate [5]. Therefore, for each server i, CPU utilization ($U_{i,t}$) is equal to $\sum_{j=1}^{N_{vm}} \sum_{k=1}^{N_c} U_{c_{(k,j,i)}}(t)$ and the power consumption of the server is estimated through Equation 2.

$$P_i(t) = \begin{cases} P_i^{idle} + (P_i^{max} - P_i^{idle}) * U_{i,t} & N_{\text{vm}} > 0 \quad (2) \\ 0 & N_{\text{vm}} = 0 \quad (3) \end{cases}$$

The energy efficiency of the consolidation algorithms is evaluated based on the data center energy consumption obtained from Equation 1.

TABLE I: Description of symbols used in Section II.

Symbol	Description
$P_{dc}(t)$	Power Consumption of the data center at time t
$P_i(t)$	Power Consumption of Server i at time t
N_s	Number of servers
P_i^{idle}	Idle power Consumption of Server i
P_i^{max}	Maximum power Consumption of Server i
$U_{i,t}$	CPU Utilization percentage of Server i at time t
N_{vm}	Number of vms
N_{c}	Number of containers
$U_{c_{(k,j,i)}}(t)$	CPU utilization of container k on (VM j, Server i) at time t
N_{v}	Number of SLA Violations
t_p	The time t at which the violation p happened
vm_{ji}	VM j on server i
$\text{CPU}_r(vm_{ji}, t_p)$	CPU amount requested by VM j on server i at time t_p
$\text{CPU}_a(vm_{ji}, t_p)$	CPU amount allocated to VM j at time t_p
$S_{(i,\text{r})}$	Server i Capacity for resource r
$U_{vm_{j,i}(t)}$	CPU Utilization of VM j on Server i at time t
$vm_{(j,i,r)}$	The capacity of resource r of VM j on server i
$c_{(k,j,i,r)}$	The resource r capacity of container k on (VM j, server i)

B. SLA Metric

Since in our targeted system we do not have any knowledge of the applications running inside the containers, definition of the SLA metric is not straightforward. In order to simplify the definition of the SLA metric, we defined an overbooking factor for provisioning containers on virtual machines which is defined by the costumer based on the percentile of the application workload at the time of the container request submission. Hence, SLA is violated only if the virtual machine on which the container is hosted on do not get the required amount of CPU that it requested. In this respect, the SLA metric is defined as the fraction of the difference between the requested and the allocated amount of CPU for each VM (Equation 4 [6]).

$$SLA = \sum_{i=1}^{N_{\text{s}}} \sum_{j=1}^{N_{\text{vm}}} \sum_{p=1}^{N_{\text{v}}} \frac{\text{CPU}_r(vm_{j,i}, t_p) - \text{CPU}_a(vm_{j,i}, t_p)}{\text{CPU}_r(vm_{j,i}, t_p)} \qquad (4)$$

C. Problem Formulation

In order to minimize the power consumption of a data center with M containers, N VMs and K servers, we formulate the problem as follows:

$$min(P_{dc}(t) = \sum_{i=1}^{N_S} P_i(t)) \qquad (5)$$

Considering the following constraints:

$$\sum_{j=1}^{N_{\text{vm}}} U_{vm_{j,i}}(t) < S_{(i,\text{r})}, \ \forall i \in [1, N_s], \ \forall r \in \{CPU\} \qquad (6)$$

$$\sum_{j=1}^{N_{\text{vm}}} vm_{(j,i,r)} < S_{(i,r)}, \ \forall i \in [1, N_s] \\ , \ \forall r \in \{BW, Memory, Disk\} \qquad (7)$$

$$\sum_{k=1}^{N_{\text{c}}} U_{c_{(\text{k,j,i})}}(t) < vm_{(j,i,r)}, \ \forall j \in [1, N_{vm}] \\ , \ \forall i \in [1, N_s], \ \forall r \in \{CPU\} \qquad (8)$$

$$\sum_{k=1}^{N_c} c_{(k,j,i,r)} < vm_{(j,i,r)},\ \forall j \in [1, N_{vm}],\ \forall i \in [1, N_s] \quad (9)$$
$$,\ \forall r \in \{BW, Memory, Disk\}$$

Although optimization toolkits can be employed to find a near-optimal solution for the above-mentioned problem, the computation time and complexity increases exponentially with the number of containers. Therefore, in section IV we evaluate set of heuristic algorithms that can obtain near-optimal solution with less computational overhead.

III. System Model

The proposed model targets a CaaS environment where applications are executed on containers. Users of this service submit requests for provisioning of containers. Containers run inside the virtual machines that are hosted on physical servers. Both physical servers and VMs are characterized by their CPU performance, memory, disk, and network bandwidth. Likewise, containers are characterized by the demand of the aforementioned resources. The objective of the system is to consolidate containers on the smallest number of VMs and consequently the smallest number of physical servers. The framework consists of 'Host Status' and 'Consolidation' modules which are shown in Figure 2 along with their components.

A. Host Status Module

The host status module which executes on each active hosts in the data center, consists of three main components as follows.

1) Host Over-load/ Under-Load Detector: The host over-load/under-load detection algorithms which will be discussed in Section IV are both implemented in this component. The component checks the resource utilization of the host every five minutes. If the host is identified as **under-loaded**, the detector sends the host ID and the IDs of all the containers running on the host to the consolidation module. This is done in an attempt to shut down the under-loaded host if the consolidation module can find new destinations for all the containers. However, if the host status is identified as **over-loaded** the detector sends a request to activate the **Container Selector** component.

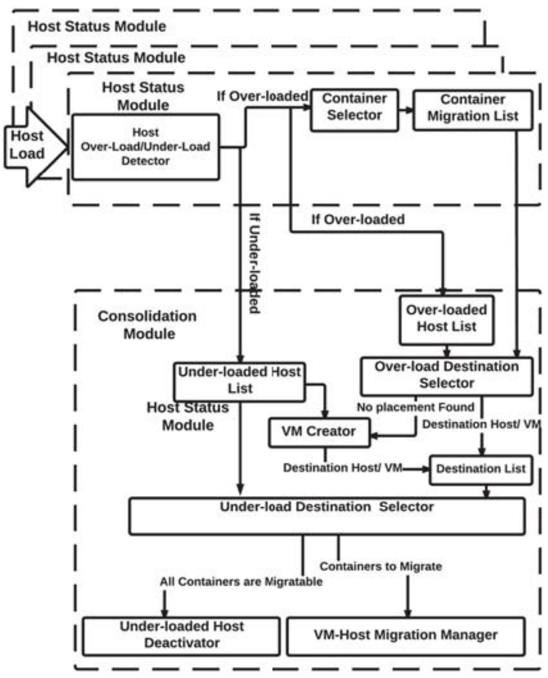


Fig. 2: System Architecture and Processes.

Algorithm 1: Overview of the Container Selector process.

```
    Input: $serverContainerList(SCL)$
    Output: $SelectedContainersList$
1   while $host\ status\ is\ Overloaded$ do
2       $container \leftarrow$
3       $ContainerSelectionAlgorithm.getContainer(SCL)$
        $SelectedContainersList.add(container)$
        $SCL.remove(Container)$
```

2) Container Selector: The container selection algorithm which will be discussed in Section IV is implemented in this component. The component is activated whenever the host is identified as **overloaded** and the container selection process continues until the host status is no longer over-loaded (Algorithm 1).

3) Container Migration List (CML): The information about containers selected by the Container Selector are saved in this component and are submitted to the consolidation module.

B. Consolidation Module

The consolidation module is installed on a separate node and can be replicated to avoid single point of failure. This module identifies an appropriate destination for the selected containers to be migrated.

1) Over-loaded Host List: This component stores hosts that are identified as over-loaded by their status.

2) Over-loaded Destination Selector: This component finds the appropriate destination for the containers in the received CMLs utilizing the host selection algorithm which will be discussed in Section IV. The process of this component is shown in Algorithm 2.

3) Destination List: Data received from the **Overloaded Destination Selector** component, which contains the container ID along with the host and the VM ID of the migration destination, are stored by this component.

4) VM Creator: This component is responsible for estimating the number of required VMs to be instantiated in the next processing window. The estimation is done based on the number of containers for which the over-load destination selector was unable to assign an appropriate host or VM as the destination. The priority of this component is creating the largest VM possible on under-loaded hosts and assigning the containers to these new VMs. If any container is left, it then chooses a random host from the inactive hosts and creates VMs on this host as long as there are no containers left without any migration destination.

Algorithm 2: Over-loaded Destination Selection process.

```
Input: $overLoadedHostList(SCL)$,
       $CMLs$,$activeHosts$
Output: $Destination(hostId, VmId)$
1  containerList.addAll(CMLs)
2  $availableHostList(AHL) \leftarrow$
   $activeHosts$.removeAll($SCL$)
3  containerList.sortByCPUUtilization()
4  foreach container in containerList do
5      $destination \leftarrow$
       HostSelectionAlgorithm.getplacement(AHL,container)
       if $destination \neq null$ then
6          $containerList.remove(container)$
7          Send $destination$ and $container.getId()$ to
           Destination List Component
8      else
9          Send $container$ to VM Creator component
10 Activate the VM Creator component.
```

5) *Under-loaded Host List:* Hosts that are found to be under-loaded by their status module are stored on this list.

6) *Under-loaded Destination Selector:* Considering the under-loaded host list and the destinations decided by the overload destination selector, this component finds the best destination for containers from the under loaded hosts (Algorithm 3) using host selection algorithm which will be discussed in the next section. If this component finds an appropriate destination for all the containers of an under-loaded host, it then sends the destination of the containers together with the destinations decided by the over-loaded destination selector to the VM-Host Migration Manager component. It also sends the host ID to the Under-loaded host deactivator component.

7) *VM-Host Migration Manager:* The containers ID together with the selected destinations are all saved in this component, and are used for triggering the migration.

8) *Under-loaded Host Deactivator:* It switches off under-loaded hosts that had all their containers migrated.

IV. Algorithms

In this section, we briefly discuss the algorithms implemented in the components of the 'Host Status' and 'Consolidation' modules of the proposed framework. As we use correlation analysis in the algorithms, we start with briefly describing the Pearson's correlation analysis.

A. Correlation Analysis

The Pearson's correlation analysis of the container load X and host workload Y performed by the selection algorithms are discussed here. This analysis results in an estimate named "Pearson's correlation coefficient" that quantifies the degree of dependency between two quantities. According to Pearson's analysis, if there are two random variables X and Y with n samples denoted by x_i and y_i the correlation coefficient is calculated using Equation 10 where $\bar{x}$ and $\bar{y}$ denote the sample means of X and Y respectively and r_{xy} varies in the range $[-1, +1]$.

Algorithm 3: Under-loaded Destination Selector process.

```
Input: $destinationList(DL)$,
       $underloadedHostList(UHL)$,$activeHosts$
Output: $ContainersToMigrateList$
1  $availableHostList(AHL) \leftarrow$
   $activeHosts$.removeAll(hosts in $DL$)
2  $ContainersToMigrateList.addAll(DL)$
3  $UHL$.sortByCpuUtilizationInDescendingOrder()
4  foreach host in $UHL$ do
5      if $host.getId()$ is in $DL$ then
6          continue
7      else
8          $AHL$.remove(host)
9          $containerList \leftarrow host.getContainerList()$
           foreach container in containerList do
10             $destination \leftarrow$ HostSelectionAlgorithm.
11                     getplacement(AHL,container)
12             if $destination \neq null$ then
13                 $containerList$.remove($container$)
14                 $tempDestList$.add($destination$)
15         if $containerList.size()$ is equal to 0 then
16             ContainersToMigrateList.addAll(tempDestList)
17             Send the $host.getID()$ to the Under-loaded
               host deactivator component.
18         else
19             $AHL$.add(host)
20 Send $ContainersToMigrateList$ to the Migrate to
   VM-Host Migration Manager component.
```

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{10}$$

The more positive and closer the correlation coefficient of X and Y gets to +1, the variables are more likely to have their peak/valley together. In other words, if the container workload is not correlated with the host load, there is less probability of that container causing the host to become over-loaded.

B. Host Status Monitor Module

1) *Overload and Under-load Detection Algorithms:* These algorithms are implemented in the *Host Over-load/Under-load Detector* component and are responsible for identifying host status. We consider static thresholds T_{ol} and T_{ul} as the criteria for a host to be over-loaded (Equation 11) or under-loaded (Equation 12) respectively .

$$\text{Host Status} = \begin{cases} \text{Overloaded} & \text{if } U_{(i,t)} > T_{ol} \quad (11) \\ \text{Under-loaded} & \text{if } U_{(i,t)} < T_{ul} \quad (12) \end{cases}$$

2) *Container Selection Algorithms:* This algorithm is implemented in the *Container Selector* component and is responsible for selecting a number of containers to migrate from the

over-loaded hosts so that the host is no longer over-loaded. The selected container list is saved in the *Container Migration List* and passed to the consolidation module to find a new VM for the containers. Here, we consider two algorithms as bellows:

- **Maximum Usage (MU) Container Selection Algorithm:** Here, the container that has the maximum CPU usage is selected and added to the migration list.
- **Most Correlated (MCor) Container Selection Algorithm:** Here, the container that is the most correlated one with the host load is chosen and added to the migration list.

C. Consolidation Module

The algorithms in this section are implemented in the consolidation module where the new destination is assigned for the CMLs received from the over-loaded hosts and the containers of the under-loaded hosts. The new destination contains the new host ID and the VM ID that the container should be migrated to.

1) Host Selection Algorithms: The host selection algorithm is implemented in the Overload and Under-load destination Selector components. The output of the algorithm contains the host and VM ID of the migration destination. The following host selection algorithms are studied in this paper. *In all the following selection methods, the virtual machine is chosen using the First-Fit algorithm based on a given percentile of the container's CPU workload.*

- **Random Host Selection Algorithm (RHS):** It selects a random host from the available host list AHL that can host the container on at least on one of its VMs.
- **First Fit Host Selection Algorithm (FFHS):** Chooses the first host of the available host list (AHL) that meets the container's resource requirements.
- **Least Full Host Selection Algorithm (LFHS):** The AHL list is sorted according to its CPU utilization in descending order. Then, the first host in the sorted AHL that meets the resource requirements of the container is selected as the migration destination.
- **Correlation Threshold Host Selection Algorithm (CorHS):** The algorithm first checks if the CPU workload history of containers and hosts are adequate for correlation analysis. In a case that the workload history is not available, it simply uses LFHS. If the workload history is available, the first host that meets the initial correlation threshold constraint and can accommodate the container on one of its VMs is chosen. If no hosts are found, the threshold is relaxed by 0.1 until a host is found (Algorithm 4).

Algorithm 4: Correlation Threshold Host Selection Algorithm.

```
Input: availableHostList($AHL$), $container$, correlation
       Threshold ($0 < thr < 1$)
Output: destination
1  $find \leftarrow false$
2  $containerloadHistory \leftarrow container.getLoadHistory()$
3  if containerloadHistory.size() is less than 5 then
       /* Find a host utilizing a substitute policy. */
4      $destination \leftarrow$ LeastFullHostSelectionAlgorithm.getHost($container$, $AHL$)
5      $find \leftarrow true$
6  while !find do
7      foreach host in $AHL$ do
8          $hostloadHistory \leftarrow host.getLoadHistory()$
9          cor ←Compute the correlation between
           hostloadHistory and containerLoadHistory
10         if $cor < thr$ then
11             if host.allocate(container) then
12                 $destination$ ←host.getId(), host.getVmId(container)
13                 $find \leftarrow true$
14                 break
       /* If no hosts are found then relax the threshold by 0.1 */
15     $cor \leftarrow cor + 0.1$
16     if $cor > 1$ then
           /* Stop the search, cor can not be bigger than 1. */
17         $destination \leftarrow null$
18         break
```

V. Performance Evaluation

A. Simulation Setup

We extended the widely used cloud simulation toolkit CloudSim [3] to model a CaaS provider. The extended package adds the support for modeling containers and their migration in a cloud data center. In our model, we consider 0.4 seconds startup delay for each container[3] and 100 seconds startup delay for VM creation [7]. These startup times are important as they directly affect the SLA metric.

A data center with 700 heterogeneous servers of three different types is simulated. Characteristics of each server together with VM and container configurations is shown in Table II. Network bandwidth is 1 GB/s, 10 MB/s, and 250 KB/s for servers, VMs, and containers, respectively. The same assumption is made for disk bandwidth and it is 1 TB, 2.5 GB, and 0.1 GB for servers, VMs, and containers, respectively.

In order to evaluate the algorithms considering the aforementioned simulation set up and configurations, we applied the workload traces from PlanetLab [4]. These traces contain 10 days of the workload of randomly selected sources from the testbed that were collected between March and April 2011 [6].

Each container is assigned to one of the workloads containing one day of CPU utilization data which is reported every 5 minutes. In order to consolidate more containers on

[3]Docker Performance Tests: http://sickbits.net/some-docker-performance-tests/

each virtual machine, a predefined (e.g 80th) percentile of the workload is considered while packing the containers on the VMs using the First Fit algorithm.

B. Experiment Results

In this section, we investigate the impact of the algorithms presented in Section IV and its tuning parameters on the system performance and data center energy consumption. Four sets of experiments were conducted with different objectives. Each experiment on each set is repeated 30 times and results are compared considering four metrics, namely SLA violations (as discussed in Section II), energy consumption, container migrations rate (number of containers migrated in each 5 minutes time slot), and average number of VMs created during the 24 hours simulation period.

Impact of the Over-load (OL) Threshold: We investigated the effect of the 'OL threshold' in the *Host Over-load/Under-load Detector* component that identifies the host status. Figure 3 shows that, for all the algorithms, increasing "OL" decreases the container migration rate since less hosts would be identified as over-loaded and less number of containers would be chosen to migrate. This decrease in the container migration rate results in less number of VM creations. On the other hand, higher 'OL threshold' increases the probability that the VMs could not get the required resources for the containers to run and this would increase the SLA violations. For "CorHs" and "FFHS," 80% is the most efficient threshold in terms of the energy consumption. For "LFHS" and "RFHS" 100% is the best option in terms of the energy efficiency, however as Figure 3 depicts average SLA violation considerably raises when compared to 80% and 90% thresholds. "CorHs" with 80% over-load threshold consumes 7.41% less energy on average when compared to other experiments with a reasonable average SLA violations (less than 5%).

Impact of the Under-load (UL) Threshold: In this set of experiments we vary the "UL" threshold while keeping the other parameters fixed, as shown in Table III. As shown in Figure 4 ,decreasing the under-load threshold increases the number of container migrations since more hosts would be identified as under-loaded as the threshold increases. Higher container migration rate results in more VMs to be created to host the migrating containers. This means more SLA violations since the container needs to wait for the VM to start-up. 70% is the most energy efficient threshold for all the algorithms except for "LFHS" because of the bigger gap between the number of the VMs created in 70% under-load threshold and the other two thresholds (Figure 4c). "CorHS" with 70% under-loaded threshold outperforms the other algorithms by 7.46% on average considering energy consumption with less than 5% SLA violations (Figure 4).

Impact of Container Selection Policies: As Figure 5 shows, since "MU" chooses the container with the biggest utilization, it requires fewer containers to migrate when the host is over-loaded and consequently results in smaller number of migrations. However, this selection increases the number of VMs required as the migration destination since most of the large containers are selected. Although the delay for starting containers is smaller than starting VMs, the higher container migration rate in "MCor" results in more SLA violations than "MU". Considering all experiments, "CorHS" is the most energy efficient host selection algorithm with 7.45% less consumption and less than 5% SLA violation.

Impact of container overbooking: Overbooking is an important factor that affects the efficiency of consolidation algorithms in terms of energy utilization and the SLA violations. Here, containers are allocated to VMs based on the predefined percentile of the application workload running on each container (Table III). The higher percentile results in smaller number of containers accommodated on each VM. Therefore, as Figure 6 illustrates 20th percentile results in fewer VMs being created and consequently less energy consumption and more SLA violations.

The number of container migrations is the same for most of the algorithms since the variance of the workload is low and migration decisions are based on the host load rather than VM load.

VI. Related Work

Unlike the extensive research on energy efficiency of computing [8], [9] and network resources [1], [10]–[12] for virtualized cloud data centers, only few works investigated the problem of energy efficient container management. Ghribi [13] discussed energy-efficient management of containers in data centers, however the effectiveness of the proposed algorithms has not been evaluated. Spicuglia et al. [**?**] proposed OptiCA, which simplifies the deployment of big data applications in CaaS. The aim of the proposed approach is to achieve the desired performance for any given power and core capacities constraints. OptiCA focuses on effective resource sharing across containers under resource constraints, while we focus on container consolidation to reduce energy consumption.

Dong et al. [14] proposed a greedy container placement scheme, the most efficient server first or MESF, that allocates containers to the most energy efficient machines first. Simulation results using an actual set of Google cluster data as task input and machine set show that the proposed MESF scheme can significantly improve the energy consumption as compared to the Least Allocated Server First (LASF) and random scheduling schemes.

Yaqub et al. [15] highlighted the differences between deployment models of IaaS and PaaS. They noted that the deployment model in PaaS is based on OS-level containers that host a variety of software services. In addition, they mentioned that, because of unpredictable workloads of software services and the variable number of containers that are provisioned and de-provisioned, PaaS data centers usually experience under-utilization of the underlying infrastructure. Therefore, the main contribution of their research is modeling the service consolidation problem as a multi-dimensional bin-packing and applying metaheuristics including Tabu Search, Simulated Annealing, Late Acceptance, and Late Simulated Annealing to solve the problem. We also modeled the container allocation problem; however, our solution focuses on application of correlation analysis and light-weight heuristics rather than on metaheuristics.

In our previous research [16], we considered CaaS cloud service model and presented a technique for finding efficient

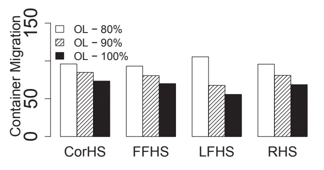
TABLE II: Configuration of servers, VMs, and containers.

Server Configurations and power models (700 Servers)					
Server type #	CPU [3GHz] (mapped on 37274 MIPS Per core)	Memory (GB)	$P^{\text{idle}}(Watt)$	P^{max} (Watt)	Population
# 1	4 cores	64	86	117	234
# 2	8 cores	128	93	135	233
# 3	16 cores	256	66	247	233

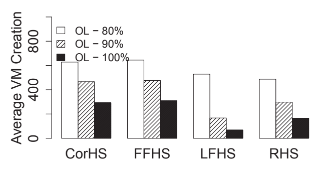
Container and VM Types (5000 Containers and 1000 VMs in total)							
Container Type #	CPU MIPS (1 core)	Memory (GB)	Population	VM Type #	CPU [1.5GHz] (mapped on 18636 MIPS Per core)	Memory(GB)	Population
# 1	4658	128	1666	# 1	2 cores	1	256
# 2	9320	256	1667	# 2	4 cores	2	256
# 3	18636	512	1667	# 3	1 core	4	256
				# 4	8 cores	8	256

TABLE III: Experiment sets, objectives, and parameters.

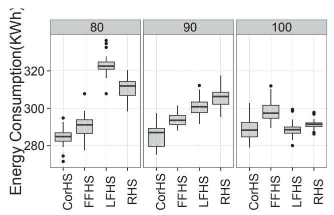
Set#	Objective	Container Selection	UL	OL	Percentile	Correlation Threshold
#1	Investigate the impact of "OL" Threshold	MU	70%	[80%, 90%, 100%]	80	0.5
#2	Investigate the impact of "UL" Threshold	MU	[50%, 60%, 70%]	80%	80	0.5
#3	Investigate the impact of container selection policies	[MU, MCor]	70%	80%	80	0.5
#4	Investigate the impact of overbooking of containers	MU	70%	80%	[20%, 40%, 80%]	0.5



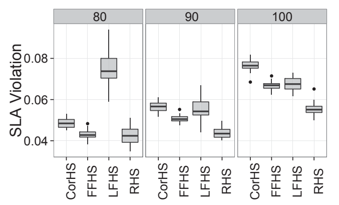
(a) Average container migrations rate per 5 minutes.



(b) Average number of created VMs.

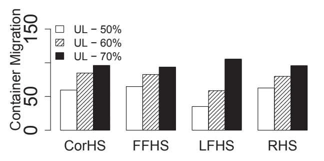


(c) Data center energy consumption.

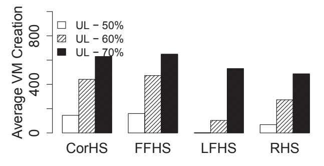


(d) SLA Violations

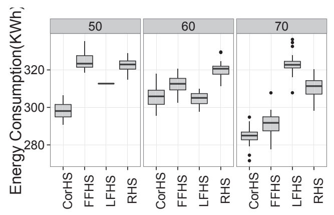
Fig. 3: Impact of over-load detection threshold "OL" on container migration rate, created VMs, data center energy consumption, and SLA violations.



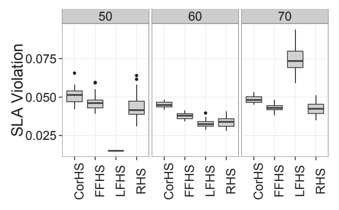
(a) Average container migrations rate per 5 minutes.



(b) Average number of VMs created during the simulation.

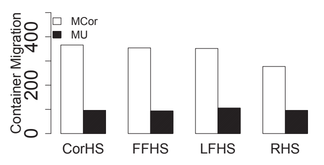


(c) Energy Consumption of the data center during the simulation.

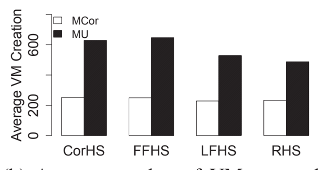


(d) SLA violations.

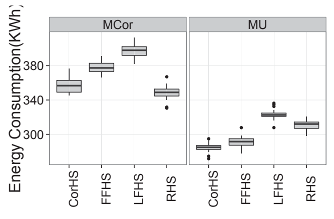
Fig. 4: Impact of under-load detection threshold "UL" on container migration rate, created VMs, data center energy consumption, and SLA violations.



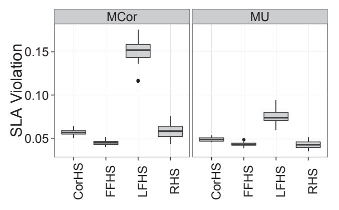
(a) Average container migrations rate per 5 minutes.



(b) Average number of VMs created during the simulation.

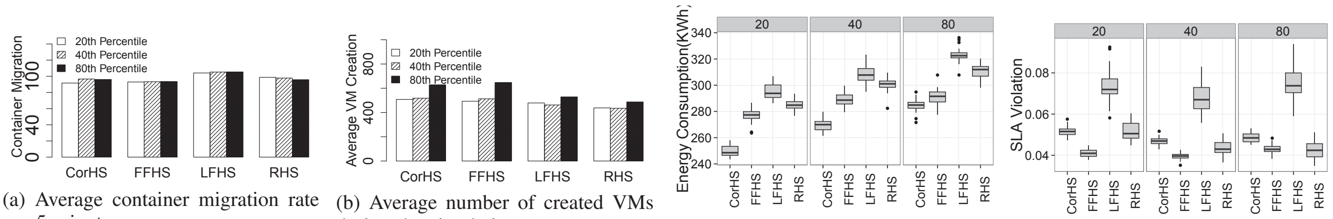


(c) Energy Consumption of the data center during the simulation.



(d) SLA violations.

Fig. 5: Impact of container selection algorithm on container migration rate, created VMs, data center energy consumption, and SLA violations.



(a) Average container migration rate per 5 minutes.

(b) Average number of created VMs during the simulation.

(c) Energy Consumption of the data center during the simulation.

(d) SLA Violations.

Fig. 6: Impact of overbooking of containers on migration rate, created VMs, data center energy consumption and SLA violations.

virtual machine sizes for hosting containers. To investigate the energy efficiency of our VM sizing technique, we considered baseline scenarios in which virtual machine sizes are fixed. Our approach outperforms the baseline scenarios by almost 7.55% in terms of the data center energy consumption. Apart from the energy perspective, our approach results in less number of VM instantiations.

VII. Conclusions and Future Work

Improving the energy efficiency of cloud data centers is an ongoing challenge that can increase the cloud providers return of investment (ROI) and also decreases the CO2 emissions which is accelerating the global warming phenomenon. Despite the increasing popularity of Container as a Service (CaaS), energy efficiency of resource management algorithms in this service model has not been deeply investigated.

In this paper, we modeled the CaaS environment and the associated power optimization problem. We proposed a framework to tackle the issue of energy efficiency in the context of CaaS through container consolidation and reduction in the number of active servers. Four sets of simulation experiments were carried out to evaluate the impact on system performance and data center energy consumption of our algorithms for triggering migrations, selecting containers for migration, and selecting destinations. Results show that the correlation-aware placement algorithm (**MCore**) with 70% and 80% as under-load and over-load thresholds outperforms other placement algorithms when the biggest container is selected to migrate (**MU**).

As future work, we will improve the placement algorithms to make them aware of VMs startup delays and migration overhead in order to decrease SLA violations.

References

[1] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *Proceedings of the 2014 IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2014, pp. 2598–2606.

[2] Q. Ali. (2015) Scaling web 2.0 applications using docker containers on vsphere 6.0. [Online]. Available: http://blogs.vmware.com/performance/2015/04/scaling-web-2-0-applications-using-docker-containers-vsphere-6-0.html

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[4] K. Park and V. S. Pai, "Comon: A mostly-scalable monitoring system for planetlab," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 65–74, Jan. 2006. [Online]. Available: http://doi.acm.org/10.1145/1113361.1113374

[5] M. Blackburn and G. Grid, Eds., *Five ways to reduce data center server power consumption*. The Green Grid, 2008.

[6] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurr. Comput. : Pract. Exper.*, vol. 24, no. 13, pp. 1397–1420, Sept. 2012.

[7] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, ser. CLOUD '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 423–430.

[8] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in *Proceedings of the Workshop on Energy-Efficient Design*, 2009.

[9] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in *Proceedings of the 2012 IEEE International Conference on Computer Communications (INFOCOM)*, March 2012, pp. 1125–1133.

[10] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 17–17.

[11] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *Proceedings of the 2012 IEEE International Conference on Computer Communications (INFOCOM)*, March 2012, pp. 2876–2880.

[12] H. Jin, T. Cheocherngngarn, D. Levy, A. Smith, D. Pan, J. Liu, and N. Pissinou, "Joint host-network optimization for energy-efficient data center networking," in *Proceedings of the 27th International Symposium on Parallel & Distributed Processing (IPDPS)*, 2013.

[13] C. Ghribi, "Energy efficient resource allocation in cloud computing environments," Ph.D. dissertation, Evry, Institut national des télécommunications, 2014.

[14] Z. Dong, W. Zhuang, and R. Rojas-Cessa, "Energy-aware scheduling schemes for cloud data centers on google trace data," in *Proceedings of the 2014 IEEE Online Conference on Green Communications (OnlineGreencomm)*, Nov 2014, pp. 1–6.

[15] E. Yaqub, R. Yahyapour, P. Wieder, A. Jehangiri, K. Lu, and C. Kotsokalis, "Metaheuristics-based planning and optimization for sla-aware resource management in paas clouds," in *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2014)*, Dec 2014, pp. 288–297.

[16] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Efficient virtual machine sizing for hosting containers as a service (services 2015)," in *Proceedings of the 2015 IEEE World Congress on Services*. IEEE, 2015, pp. 31–38.