Joint Host-Network Power Scaling with Minimizing VM Migration in SDN-enabled Cloud Data Centers

Tuhin Chakraborty*, Adel N. Toosi*, Carlo Kopp*, Peter Stuckey* and Julien Mahet†
*Faculty of Information Technology, Monash University, Clayton, Australia
†Enseirb-Matmeca (Bordeaux INP), France

***Abstract*—In recent times, both industry and academia have paid significant attention to the power management of cloud data centers (CDCs), due to their typically very high electrical energy consumption. While servers remain the components with the highest power-consumption, network stacks can also consume about 10-20 percent of the total energy usage in a data center. Dynamic Virtual Machine (VM) consolidation is one way to reduce the number of active servers, which can be done by live migration of the VMs. But, migration operations in a data center bring several system and service level overheads that include downtime, elephant flows over the network, and potentially higher failure rates. In this work, we propose algorithms for minimizing the number of VM migrations to attain the optimized joint host-network power consumption in a cloud data center. We present a trade-off between the number of migrations, the joint host-network power consumption, and the computational time complexity of the proposed algorithms. Using Mininet and ONOS, an SDN enabled framework is utilised to evaluate the proposed algorithms. Experimental results show that our algorithms can reduce power consumption by about 11 percent, while completing between 18 to 25 percent more VM migrations compared to the baseline algorithm, which only minimizes migration without guaranteeing lowest power consumption.**

I. Introduction

Cloud computing, with its inherent flexibility, elasticity, and pay-as-you-go based funding model has seen a sustained increase in popularity across both commercial and academic user populations. The consequence of this popularity has been an increase in the number and size of Cloud Data Centers (CDC) required to provide the required computing resources. Consequently, a tremendous amount of electrical energy is consumed by these data centers, this producing considerable impact on the environment globally. To be explicit, CDCs have been estimated to consume around 140 billion kWh annually up to 2020 [25], estimated to consume around 8000 TWh annually by 2030 [10].

To address this problem, there has been a sustained focus on efficiently managing the power usage of the most power hungry components of CDC infrastructure, specifically cooling systems [1], [2] and servers [3], [4], [5], [6], [7], [8]. This focus has led to energy reduction algorithms, and strategies for scalable energy utilization.

Current CDCs are designed and operated with over-provisioned resource capacity to meet peak demand performance needs with fluctuating workloads, resulting in turn in under-utilization of resources most of the time. While different power efficient algorithms for servers have previously been explored, such as placements of virtual machines (VMs) in proper hypervisors, consolidation of VMs, and migration of VMs [30], these have not always produced proportionate power reduction outcomes. This is because these measures did not always account for power consumed by network component placements where traditional CDC distributed router-based network systems are employed.

No differently to servers, CDC fabrics, typically implemented as networks, are also designed for peak demand. Network components such as switches, ports, and links are always kept powered up to maximise aggregate fabric bandwidth for peak demand periods. The advent of Software Defined Networking (SDN) removes this impediment, as it is now possible to manage network stacks in a CDC through software. Consequently network consolidation, and scaling of active network components in use is now possible. The era of forwarding by physically configuring network hardware components like switches and routers at the individual level, is becoming obsolete, instead the software controller of the SDN acts as the link between application and infrastructure layers and can isolate the forwarding plane from the network control plane. This way, it can handle network management and routing through a software controller thus enabling quick and automatic responses to the varying workload demands arising from diverse application requirements.

Researchers have also proposed power scaling-down algorithms for network stacks under low traffic conditions. Heller et al. [11] introduced the idea of the ElasticTree, which can dynamically reconfigure different network elements with the changing of traffic loads in a data-center. Datacenter topologies are being designed to ensure reliability and quality of service (QoS) during massive traffic loads arising with peak workloads. As a result, a large number of redundancies exist in CDC topologies, as these network resources are unused outside peak load periods. One can reduce the number of links by network traffic consolidation, and some network components which are not in use can be temporarily put into a dormant state as an energy-saving measure. Most power consumption in a network arises from its switching components while turned on [11]. Turning off only unused ports saves very little energy, at best 1-

2 Watts [11]. The concept of ElasticTree starts with this observation. This approach consists of finding the minimal subset of network elements that are sufficient for the CDC to maintain required internal connectivity. The ElasticTree concept did not consider hypervisors and VMs and did not switch off edge switches. Hence it still suffers from some purposeless power consumption for both its computing and network stacks.

On the other hand, different power scaling algorithms are being used that push non-active hypervisors into a dormant state. However, after consolidating active VMs into fewer servers, these algorithms only provide a limited advantage, unless they are aware of the position of edge switches connected to servers because it is not only the number of servers but also, the position of the servers that can impact joint host-network power scaling. In addition to this, migrating VMs to pack them into fewer hosts leads to the well-known adverse effect of VM migration overhead, so it is also desirable to keep the number of migrations to a minimum while seeking to attain the intended scaled-down power state of the CDC. In this work we focus on:
Jointly minimising the number of hosts, switches and VM migrations to attain the power minimised state for a given CDC workload by exploiting an SDN controller;
and we make three main contributions, as outlined below:

- Identification and explanation of the trade-off between power consumption improvement and the number of migrations during the application of the power scaling operation.
- Devising a set of three new algorithms to minimize the number of migrations and optimize power consumption, the first one focusing on the number of migrations, the second focusing on a balance between the power consumption and the number of migrations, but with the penalty of higher computational cost; and the third which is a heuristic-based algorithm, that can also balance between the two.
- Evaluation of the proposed algorithms using a new simulation tool built for this purpose. Our proposed simulation tool emulates CDC networking using the mid-level APIs of Mininet, integrated with the extended ONOS network controller. It uses Python scripts to simulate VMs, hosting servers, VM-to-Host mappings, VM migration, and power modelling of compute and network stacks of a CDC.

The rest of the manuscript is organized as follows. We explain the preliminaries of the CDC architecture used, and problem modeling in section 2. Section 3 explains our framework and its components followed by the algorithms proposed to reach down to the lowest possible power state from any given state with different VM distribution examples in 4. Section 5 explains the evaluation of results followed by related work in section 6. Finally, we conclude this work and propose further research directions in section 7.

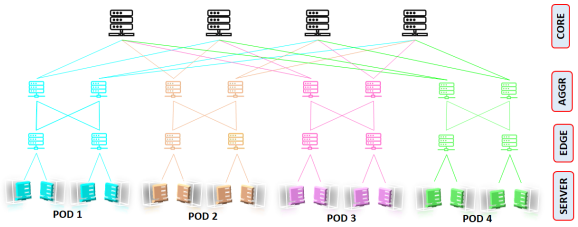


Figure 1: $K = 4$ Fat-tree Topology.

II. Problem Modeling

This section explains the problem of agile scaling down of the power consumption of a cloud data center, where the minimum number of servers and network components remain active and the number of VM migrations from the current state is minimized. For the sake of better readability, we divide this section into three subsections as follows. In section II-A, we explain data center infrastructure. We considered fat-tree-based architecture, a well-known and widely used data center architecture. In section II-B, we report the power consumption model we use in this work for the servers and network components. Thirdly, in section II-C, we formulate the problem of agile scaling down of the total power with the minimum number of migrations from any current state of VM distribution among the servers.

A. Preliminaries

We report some basic properties of widely used fat-tree DCN topology which we make use of. The concepts investigated here are easily extended to the other DCN topologies. However, the trade-offs and limitations of different data center architectures will be different, so the results presented here are not directly applicable. In brief, *fat-tree* has the same bandwidth at all bisections, with the same aggregated bandwidth at each layer; and this can provide certain benefits: it can be assembled exploiting cheap and uniform capacity devices and it is highly scalable ($K^3/4$ servers are connected with each K port edge switch). Figure 1 shows an example of $K = 4$ fat-tree topology.

In fat-tree of order K, all switches have K ports. It is a 3-layer DCN topology consisting of edge, aggregation, and core layer switches. A fat-tree of order K, consists of K pods, each pod consists of $(K/2)^2$ servers, each pod have 2 layers of $K/2$ number of switches, each edge switch is connected with $K/2$ servers and with $K/2$ aggregation switches in the same pod, each aggregation switch is connected to $K/2$ edge switches in the same pod and $K/2$ core switches at the upper layer. A fat-tree of order K has $(K/2)^2$ core switches, each connects to all K pods in the DC. A K-array fat-tree contains $(K^3)/4$ servers; K pods $\times$ $K/2$ edge switches per pod $\times$ $K/2$ servers per edge switches = $(K^3)/4$ total number of servers in the CDC.
Next, we present the basics of the power consumed by the servers and network components used in this work.

Table I: Notations

Parameter	Description
h_i	The i^{th} host in the CDC
H	The set of all hosts in the CDC, $\forall i, h_i \in H$
s_i	The i^{th} switch in the CDC
$P(h_i)$	Power consumption of host *i*
P_{idle}	Idle power consumption of host
P_{peak}	Peak power consumption of host
u_i	CPU utilization percentage of host *i*
α_i	The number of VMs placed in host *i*
$P(s_i)$	Power consumption of switch *i*
P_{sw}	Power consumption of a switch
$\|VM\|$	The total number of VMs in the DC at any given point of time
$\|H\|$	The total number of hosts in the data center
N_{vm}^{mig}	The total number of VM migrations
$\|S\|$	The minimal number of network switches required to serve the current VM load
v	The highest number of VM possible in a host
H_{min}	The minimum number of hosts required to conte--nts $\|VM\|$
E_{min}	The minimum number of edge switches required to support $\|VM\|$
A_{min}	The minimum number of aggregation switches required to support $\|VM\|$
C_{min}	The minimum number of core switches required to support $\|VM\|$
$\|S\|$	The minimum number of total switches required to support $\|VM\|$ in the DCN
N	The set of natural numbers

B. Power Model

The notations used for the problem formulation are given in Table I. We use the CPU power utilization model for the hosts from [9]. This model is also being used in other recent work [25], [26]. The idea is that a server consumes roughly half its peak-load power when it remains idle, and the power consumption grows linearly with the amount of utilization. Power can be reduced if the workload of a host can be migrated to another host, and the current one can be put into dormant state. So, the power consumption of host i can be modeled as:

$$P(h_i) = \begin{cases} P_{idle} + (P_{\text{peak}} - P_{\text{idle}}) \cdot u_i & \text{if } \alpha_i > 0 \\ 0 & \text{if } \alpha_i = 0 \end{cases} \quad (1)$$

where P_{idle} and P_{peak} are the constant factor and peak power consumption of hosts, u_i is CPU utilization percentage, and α_i is the number of VMs placed in host i.

The main power required by the network is its switch components when they are on, and switch power consumption is not proportional to the traffic that is transferred [11], so we simply count how many switch components must be powered ON to maintain connectivity in the scaled down state. Hence, the power consumption of any switch s_i (the i^{th} switch in CDC) is modeled as:

$$P(s_i) = \begin{cases} P_{\text{sw}} & \text{if } s_i \text{ is on} \\ 0 & \text{if } s_i \text{ is off} \end{cases} \quad (2)$$

where P_{sw} is power consumption of a switch.

C. Problem Formulation

The power scaling down of a CDC to a desirable power consumption state from an arbitrary state of VM distribution constitutes a trade-off between the joint number of active servers-switches and the minimum number of migrations.

Firstly, we minimize power consumption by the servers, by bin packing all the VMs present in the CDC into the minimum number of servers. However, that packing can not ensure the minimum number of switch components required to keep the data center connected in the power scaled state. In addition, awareness about the position of servers in the DCN is required to ensure the minimum network stacks. Hence, finding the proper set of servers is essential to fulfill the joint server and network power optimization goal. Furthermore, we may find such sets, but which of these can ensure the minimum VM migration to reach the desired power state? To investigate this gap, we merge all these requirements together and explore this problem and its solution with proper example scenarios and results in section IV and V.So, this problem focuses on scaling-down the energy consumed within a CDC jointly by minimizing the number of active servers and network switches to the optimized power consumption from any current VM distribution $\{D_{vm}\}$.

Let us assume, at any point of time, $|VM|$ is the total number of VMs that are present and placed in the total number of hosts $|H|$ in the CDC taking the form of a fat-tree DCN topology of order K. Then the minimum number of hosts required to accommodate these VMs is

$$H_{\min} = \left\lceil \frac{|VM|}{v} \right\rceil \quad (3)$$

where v is the highest number of VMs that can be accommodated in a single host, v is considered to be a fixed value for the sake of simplicity. The minimum number of edge switches required to be active in the DCN is –

$$E_{min} = \left\lceil \frac{H_{min}}{\frac{K}{2}} \right\rceil \quad (4)$$

where $K/2$ is the number of servers that are connected with each edge switch in fat-tree DCN. The minimum number of aggregation switches required in the DCN is –

$$A_{min} = \left\lceil \frac{E_{min}}{\frac{K}{2}} \right\rceil \quad (5)$$

The min number of core switches C_{min} required can be taken as 1, as each core switch connects to all K pods in the DC. The total number of switches needed in the DC is:

$$|S| = E_{min} + A_{min} + C_{min} \quad (6)$$

Now, our problem can be defined as: find a set C of cardinality H_{min} from the set of hosts $\{H : H \in N, H \in [1, |H|]\}$, which minimizes the objective:

$$\textbf{minimize:} \sum_{i=1}^{|H|} P(h_i) + \sum_{i=1}^{|S|} P(s_i) \quad (7)$$

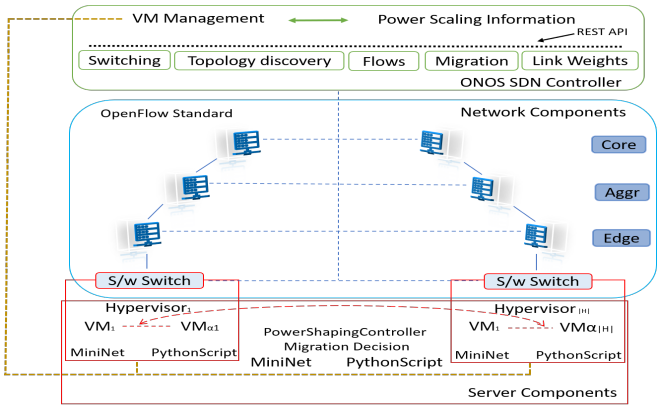


Figure 2: High-level System Architecture.

$$\text{and } \textbf{minimize: } N_{vm}^{mig} \tag{8}$$

subject to constraints:

$$\sum_{i=1}^{|H|} \alpha_i = |VM| \tag{9}$$

$$\forall i \text{ such that } i \in N, i \in [1, |H|],\ \alpha_i \leq v \tag{10}$$

where N_{vm}^{mig} is the number of VM migrations needed to reach to the optimized state. We analyze the *Power Reduction Improvement* while migrating to achieve the power optimized state.

III. Energy Scale-down Framework

To overcome the challenges of (a) selecting the H_{min} hosts from the set of hosts {H}, which are sufficient to serve the current processing load, (b) transferring the load into that selected H_{min} hosts, and (c) reaching a state with the minimum number of VM migrations — an SDN framework, along with a VM manager, and an energy calculator has been implemented to manage and support VMs among the hosts with the primary objective of realizing the step-wise power reduction improvement with the VM migration events. The proposed framework can be imagined as a marriage between an SDN and a VM manager with an energy calculator through the nexus of a centralized controller. They are working together, passing information to each other to plan and make decisions to ultimately achieve the objective of scaling the power consumption of a data center. Figure 2 represents the high-level system architecture of our proposed system to measure the trend of power reduction. The logically centralized software controller can maintain a global view of VM placements as well as the status of the servers and their placements under the connecting edge switches. As a result, it can decide upon the migration of VMs from source to destination servers, which creates the scope to reduce power consumption, not only due to the changes at the servers but also changes at the active switches. The following system components and attributes are worth elaborating on, although not all of them have similar significance in realizing our contribution, they are all necessary to build our entire emulated cloud data center architecture.

***Data Center Network and Topology*:** The data center network (DCN) is implemented using the mid-level API of Mininet [16]. We wrote a Python script to generate a fat-tree data center network as it has several beneficial properties described in section II-A, and is widely used in practice. Our Python script generates a homogeneous fat-tree, that can be defined with a full-bisection-bandwidth topology with three layers of similar network switches with k ports. The details about the K-array fat-tree topology and its properties are already reported in section II-A. Then, the external SDN controller (ONOS [17]) is added to enable the network control through the OpenFlow protocol. Although we implemented it to enable a fully fledged data center structure, our main contribution is achieved through server virtualization, and the power calculating applications.

***Server Virtualization*:** The API of Mininet [16] only can provide the servers via assigning IPs to the servers. We extended these solely IP-based servers by expanding the features incorporating simulated virtualized characteristics through a python script. We extended it so that it can simulate — the creation of VMs under the servers, maintaining VM-server mapping, and VM migrations from one server to another by making decisions based on the information coming from the other modules. Based on the popularly used power models [9], [11], our power-calculator can show improvements in power consumption arising from scaling and can receive migration information from this controller. With the cooperation of these modules, the logically centralized software controller can choose the source and destination servers for VM migrations to create a scope to reduce power consumption, not only by the servers but also by the switches. Thus, our simulated framework virtualizes the data center environment with its computing and network stacks and measures the power consumption by the server and network stacks.

In the next section, we present three algorithms to scale the power jointly consumed by servers-networks in a CDC.

IV. Energy Scaling Strategies

Reducing the power consumption in a cloud data center can be achieved by simply putting the unused resources in a dormant state. We aim to manage DC resources to retain a viable service yet reduce power consumption. This section presents the power scaling algorithms that focus on keeping the number of migrations to a minimum. It also compares the power consumption improvement, consumed jointly by server and network stacks with the changes in VM distributions when they are aware of DCN topology, including network switch positioning. The next part presents and discusses three such algorithms that can reduce the power consumption of CDCs.

$MinMig_{algo}$ can be thought of as a baseline strategy. It gives the highest priority to the number of VM migration over all the other factors. In $MinMig_{algo}$, first we calculate

Algorithm 1 $MinMig_{algo}$ algorithm

Input: K: The order of the DCN topology
Input: C: The highest number of VMs possible for each host, considering a homogeneous configuration
Input: L_h: A list the VMs under each host
Input: L_E: A list the VMs connected with each edge layer switch
Input: L_P: A list the VMs contained under each pod
Output: $Destination\#$: A list of hosts which will be receiving VMs from the other hosts during Migration for scaling down the energy in the DC
Output: $Source\#$: A list of hosts from which all the VMs will be migrating to the $Destination\#$ list during scaling down the energy in the DC

1: $H \leftarrow (K^3)/4$ ▷ The total number of servers in the CDC
2: $Destination\#, Source\# \leftarrow \theta$ ▷ List initiated to $NULL$
3: $T_V \leftarrow sum(L_h)$ ▷ Sum of all elements in L_H
4: $T_H \leftarrow \lceil \frac{T_V}{C} \rceil$ ▷ Minimum number of hosts required to accommodate all the present VMs
5: $L_S \leftarrow argReverseSort(L_h)$ ▷ Returns the indices of the reverse sorted array so that the indexed values would be reverse sorted, we required this as the index values can represent the VM numbering/labeling in DC
6: **while** $len(Destination\#) \neq T_H$ **do**
7: $Item \leftarrow L_S.pop(First\_Index)$ ▷ Return and remove the first item from L_S, assign it to $Item$
8: $Destination\#.append(Item)$
9: **end while**
10: $Source\# \leftarrow L_S$
11: **return** $Destination\#, Source\#$

how many hosts are required to pack all the VMs present in the servers (T_H, steps 1-4). Then we find that T_H number of servers from all the hosts that are currently accommodating the highest number of VMs, and we tag them as *Destination#* hosts (steps 5-9). All other remaining hosts are tagged as *Source#* hosts. Now, all VMs from the *Source#* hosts are migrated to the *Destination#* hosts (steps 10), thus ensuring the minimization of migrations. The power models in our controller can calculate the power before and after the migration event happened. However, it cannot guarantee that it can scale down the energy to the minimum level although it will require a minimum number of servers, since it may require many more network components to connect them.

$Comb_{algo}$ is based on the combinatorial selection of the hosts where the highest priority is given to the power reduction over the number of VM-migrations. Unlike $MinMig_{algo}$, it assures scaling-down the power to its mini-

Algorithm 2 $Comb_{algo}$ algorithm

Input: K: The order of the DCN topology
Input: H: Set of all hosts in the DC
Input: C: The highest number of VMs possible for each host, considering a homogeneous configuration
Input: L_h: A list of the VMs under each host
Input: L_E: A list of the VMs connected with each edge layer switch
Input: L_P: A list of the VMs contained under each pod
Output: $Destination\#$: A list of hosts which will be receiving VMs from the other hosts during Migration for scaling down the energy in the DC
Output: $Source\#$: A list of hosts from which all the VMs will be migrating to the $Destination\#$ list during scaling down the energy in the DC

1: $h \leftarrow (K^3)/4$ ▷ The total number of hosts in the CDC
2: $Destination\#, Source\#, R, L^S_{Comb} \leftarrow \theta$ ▷ List initiated to $NULL$
3: $T_V \leftarrow sum(L_h)$ ▷ Sum of all elements in L_h
4: $Max \leftarrow 0$ ▷ Max value initiated to 0
5: $T_h \leftarrow$T$_h \leftarrow \lceil \frac{T_V}{C} \rceil$ ▷ Minimum number of hosts required to accommodate all the present VMs
6: $T_S \leftarrow minNumberOfSwitchNeeded$ By Eqation 6 ▷ Minimum number of switches to support all the present VMs
7: $L_{Comb} \leftarrow set(list(combinations(H, T_h)))$ ▷ list of all unique subsets of length T_h taken from H
8: **for** each subset_Item S in L_{Comb} **do**
9: **if** $minNumberOfSwitchRequiredfor$**S** = T_S **then**
10: **if** $Count\_Number\_Of\_VMs\ in$ **S** $< Max$ **then**
11: $Max \leftarrow Count\_Number\_Of\_VMs\ in$ **S**
12:
13: $R \leftarrow list$(**S**)
14: **end if**
15: **end if**
16: **end for**
17: $Destination\# \leftarrow R$
18: $Source\# \leftarrow list(\{H\} - \{R\})$
19: **return** $Destination\#, Source\#$

mum possible level. Additionally, $Comb_{algo}$ selects a set of $Destination\#$ hosts that accommodate the highest number of VMs to ensure the lowest amount of VM migration. In $Comb_{algo}$, first we calculate how many hosts are required to pack all the VMs present in the servers (T_h, steps 1-5). Then we make a list L_{Comb} of all unique subsets of length T_h (step 7) taken from all the hosts ($h = K^3/4$). Then from L_{Comb}, we compute a single set of hosts (R, step 13), that uses the minimum number of switches to connect (to ensure

Algorithm 3 Heu_{algo} algorithm

Input: K: The order of the DCN topology
Input: H: Set of all hosts in the DC
Input: C: The highest number of VMs possible for each host, considering a homogeneous configuration
Input: L_H: A list of the VMs under each host
Output: $Destination\#$: A list of hosts which will be receiving VMs from the other hosts during Migration for scaling down the energy in DC
Output: $Source\#$: A list of hosts from which all the VMs will be migrating to the $Destination\#$ list during scaling down the energy in DC

$H_n \leftarrow (K^3)/4$ ▷ The total number of servers in the CDC
$Destination\# \leftarrow \theta$ ▷ List initiated to $NULL$
$Source\# \leftarrow \theta$ ▷ List initiated to $NULL$
$L_E \leftarrow \theta$ ▷ List of number of VMs connected with each edge layer switch initiated to $NULL$
$L_P \leftarrow \theta$ ▷ List of number of VMs contained under each pod initiated to $NULL$
$T_V \leftarrow sum(L_H)$ ▷ Sum of all elements in L_H
$T_H \leftarrow \lceil \frac{T_V}{C} \rceil$ ▷ Minimum number of hosts required to accommodate all the present VMs
$temp \leftarrow 0$ ▷ Temporary value initiated to 0
for $Iteration$ **i** $= 0, 1, \ldots, K^3/4$ **do** ▷ As the total number of servers is $H_n = K^3/4$
  if i $modulo$ **K/2** = 0 **then** $L_E.Append(0)$
  end if
  if i $modulo$ $K^2/4$ = 0 **then** $L_P.Append(0)$
  end if
  L_E[int$(i/(K/2))$] += L_H[i] ▷ List of number of VMs connected with each edge switch
  L_P[int$(i/(K^2/4))$] +← L_H[i] ▷ List of number of VMs contained under each pod
end for
$L_P^{RevS} \leftarrow argReverseSort(L_P)$
if $\left\lfloor \frac{T_V}{C \times K^2/4} \right\rfloor \neq 0$ **then**
  for $Iteration$ **i** $= 1, 2, \ldots, \left\lfloor \frac{T_V}{C \times K^2/4} \right\rfloor$ **do**
    $tempPodNumber \leftarrow L_P^{RevS}.pop(First\_Index)) + 1$
    for each host **h** in $tempPodNumber$ **do**
      $Destination\# \leftarrow Destination\#.append(h)$
    end for
  end for
end if
if T_H $modulo$ $K^2/4 \neq 0$ **then** ▷ If any partially full pod needed
  $P_{Part}^{Num} \leftarrow L_P^{RevS}.pop(First\_Index)) + 1$ ▷ Pod serial number in the DC of this partially full Pod
  N_{Part}^P = T_H $modulo$ $K^2/4$ ▷ number of host for this partially full Pod
  if $\left\lfloor \frac{N_{Part}^P}{K/2} \right\rfloor \neq 0$ **then**
    $L_E^{Part} \leftarrow L_E[((P_{Part}^{Num} - 1) \times K/2) : ((P_{Part}^{Num} \times K/2) - 1)]$ ▷ list[n : m] ← return a sub-list from index n to m
    $L_{EPart}^{RevS} \leftarrow argReverseSort(L_E^{Part})$
    for $Iteration$ **i** $= 1, 2, \ldots, \left\lfloor \frac{N_{Part}^P}{K/2} \right\rfloor$ **do**
      $temtE_{Part} = L_{EPart}^{RevS}.pop(First\_Index)$
      for each host **h** in $L_E[((P_{Part}^{Num} - 1) \times K/2) + temtE_{Part} + 1]$ **do**
        $Destination\# \leftarrow Destination\#.append(h)$
      end for
    end for
  end if
end if
if N_{Part}^P $modulo$ $K/2 \neq 0$ **then** ▷ Selection of lone VMs
  $temp_H^{Num} = N_{Part}^P \bmod K//2$
  $tempE_{Part} = L_{EPart}^{RevS}.pop(First\_Index)$
  $Left_{Idx} = ((P_{Part}^{Num} - 1) \times K^2/4) + ((tempE_{Part} - 1) \times K/2)$
  $Right_{Idx} = ((P_{Part}^{Num} - 1) \times K^2/4) + (((tempE_{Part} - 1) \times K/2) + k/2 + 1)$
  $temp_H^{list} = L_H[Left_{Idx} : Right_{Idx}]$
  $L_{HPart}^{RevS} \leftarrow argReverseSort(temp_H^{list})$
  for $Iteration$ **i** $= 1, 2, \ldots, temp_H^{Num}$ **do**
    $tempH_{Part} = L_{HPart}^{RevS}.pop(First\_Index)$
    $h = Left_{Idx} + tempH_{Part}$
    $Destination\# \leftarrow Destination\#.append(h)$
  end for
end if
$Source\# \leftarrow H - Destination\#$
return $Destination\#, Source\#$

the joint host-network power optimization), and accommodates the highest number of VMs (to ensure minimum VM migrations) (step 8-16). Power is prioritized over migration (step 9-10). Then, this set of hosts are tagged as *Destination#* hosts, and the remaining hosts are tagged as *Source#* hosts (step 17-18). Now, all VMs from the *Source#* hosts will be migrated to the *Destination#* hosts. The power model of our controller can calculate the power before and after the migration event occurs.

Importantly, the time complexity of enumerating all subsets of size *r* from a set of *n* elements using an iterative method is $\Theta(n \times 2^n)$, which is exponential and not tractable. For example, if *n*=50 and *r*=2, it requires roughly 10^{15} passes to iterate through combinations to produce only 1225 results [15]. If we use recursive calls, it takes $O(n \times nC_r)$ as an upper bound on the number of recursive calls [15], and in our case, this is $O\left(K^3/4 \times_{K^3/4} C_{T_h}\right)$, where K is the order of fat-tree, and T_h is the minimum number of hosts required. The excessive time complexity of this approach makes it unattractive for practical implementation. Hence, we next present a heuristic-based host selection mechanism to address this issue.

Heu_{algo} is based on the heuristic selection of the hosts from the pods, which contain the highest number of VMs. This algorithm makes use of the DCN topology when selecting the hosts. Here, first, we calculate the number of hosts needed to pack all the VMs in the server (T_H, step 1-7). Then we calculate the minimum number of entire pods (where all hosts are needed to be active) required to accommodate all the hosts; let say it is P ($P = \left\lfloor \frac{T_V}{C \times K^2/4} \right\rfloor$). Steps 9-16 compute the list L_P and L_E, containing the list of the numbers of the VMs accommodated under each pod and number of VMs connected under each switch, respectively. Steps 17-25 appended all hosts from P number of entire pods needed first and tagged them as $Destination\#$ hosts.

At this point, it selects a partial pod, and the number of partial pods required can be either zero or one. If it is one, the algorithm checks the members of L_E, which belongs to that partial pod only (L_E^{Part}, step 30), and sorts them in the reverse order(L_{EPart}^{RevS}, step 31). Then, it counts the minimum number of full edge switches required. Full edge switch means – all the hosts connected to this edge switch is selected as $Destination\#$ hosts (steps 26-39). After selecting full edge switches, the next edge switch from L_{EPart}^{RevS} is selected as a partial edge switch as it connects the next highest number of VMs. This step happens if the number of hosts already tagged as $Destination\#$ still not equal to T_H, the minimum number of hosts needed. Then the remaining number of hosts required to tag as $Destination\#$ is selected from this edge switch. The remaining number of hosts are the highest number of VM accommodating hosts connected to this edge switch and is tagged as $Destination\#$ (steps 40-52). All other remaining hosts are tagged as *Source#* hosts (step 53). Now, all VMs from *Source#* hosts are migrated to the *Destination#* hosts. The power estimating model of our controller can calculate the power before and after the migration event occurs.

Importantly, the number of enumerations in Heu_{algo} is practically small and much less than in $Comb_{algo}$. Once we sort the list L_P, containing the number of VMs accommodated under each pod, it requires at most K (K number of pods exist in K-order fat-tree) iterations. Similarly, it requires at most $K/2$ iterations to select the full edge switches from the partial pod, and again requires at most $K/2$ iteration to select the remaining number of hosts to become a member of the $Destination\#$ hosts. Once we consider the required sorting operations before running each iteration mentioned above, then the computation cost will be — (1) for selecting hosts to be a member of $Destination\#$ from *Full Pods*: $\mathcal{O}(K \times \log K)$, for sorting; $\mathcal{O}(K)$, for rolling over iterations; (2) for selecting hosts from partial pod: (a) selecting hosts to be a member of $Destination\#$ from *Full Edges* of the *Partial Pod*: $\mathcal{O}(K/2 \times \log K/2)$, for sorting; $\mathcal{O}(K/2)$, for rolling over iterations; (b) selecting remaining hosts to be a member of $Destination\#$ from *Non-Full Edge* of the *Partial Pod*: $\mathcal{O}(K/2 \times \log K/2)$, for sorting; $\mathcal{O}(K/2)$, for rolling over iterations; In brief the total computation cost will be ($\mathcal{O}(K \times \log K)$ + $\mathcal{O}(K)$) + ($\mathcal{O}((K/2) \times \log(K/2))$ + $\mathcal{O}(K/2)$) + ($\mathcal{O}((K/2) \times \log(K/2))$ + $\mathcal{O}(K/2)) \equiv \mathcal{O}(K \times \log K)$ + $\mathcal{O}(K)$.
The next section discusses the improvements in the power consumption with the triggering of consecutive migrations throughout the scaling process in a simulated environment.

V. Performance Evaluation

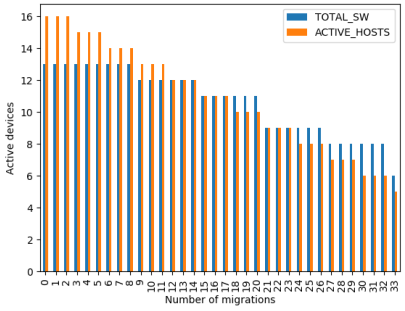
We evaluate the proposed algorithms in a simulated environment described in section III. We measure the energy reduction rate with the triggering of each migration event during the energy scaling procedure. Each migration event in the data center may change the total energy consumption of the center, so, we measure the total energy consumption before and after each migration event occurs, to measure the effect of the most recent migration. We compare the numbers of VM migrations taken by each of the algorithms to push the data center towards a minimum power consumption state.

A. Experimental Setup

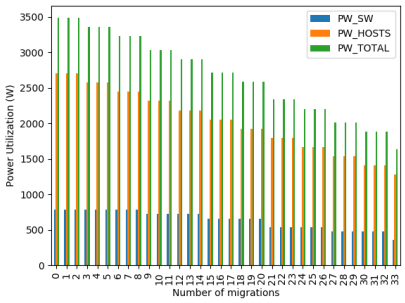
The data center simulated for this experiment employs a $K = 4$ fat-tree topology, with 16 hosts, each with ten processing units. We assume that each host has sufficient main memory and storage capacity, and the primary decision making measure is the CPU power consumption. Power consumption of the CDC is measured, by the power values of servers and switches [9] using the power models in [9], [11] as described in section II-B. We ran these simulations on Intel Core i7-8850H 2.6GHz $\times$ 12 CPU, 64-bit computer with 32 GB RAM, running Ubuntu 18.04.3.

Table II: Initial VM distributions in servers used in the experiments for $K = 4$ fat-tree cloud data center

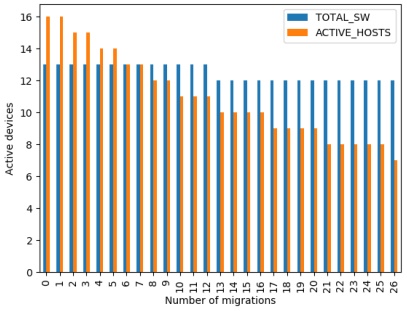
Exemplary VM Distribution Types Used for This Work	VM Distribution Under the Servers and how They are Connected Under the DCN															
	Pod1				Pod2				Pod3				Pod4			
	E1		E2		E3		E4		E5		E6		E7		E8	
	VM_1	VM_2	VM_3	VM_4	VM_5	VM_6	VM_7	VM_8	VM_9	VM_{10}	VM_{11}	VM_{12}	VM_{13}	VM_{14}	VM_{15}	VM_{16}
Type#1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Type#2	2	5	6	2	7	4	2	5	5	2	4	7	7	5	2	3
Type#3	2	2	2	7	5	5	4	5	5	7	6	2	3	3	7	5



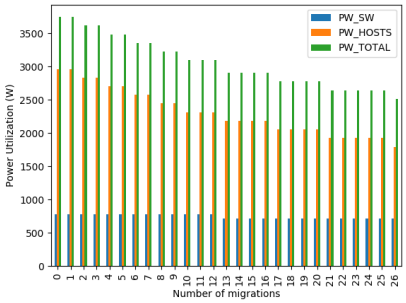
(a) Number of active devices changes with migration for *Type#1* initial placement, all three algorithms produced the same results



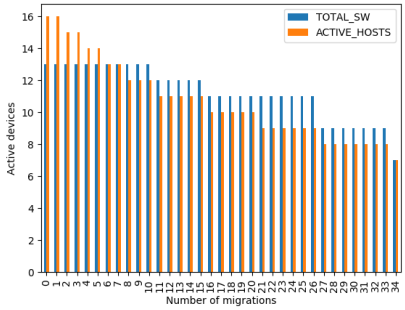
(b) Power utilization by network and computing stacks changes with migration for *Type#1* initial placement, all three algorithms produced the same results



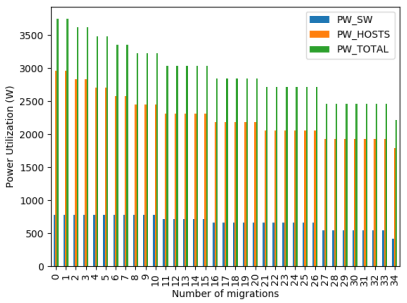
(c) Number of active devices changes with migration for *Type#2* initial placement in $MinMig_{algo}$



(d) Power utilization by network and computing stacks changes with migration for *Type#2* initial placement in $MinMig_{algo}$



(e) Number of active devices changes with migration for *Type#2* initial placement in $Comb_{algo}$ and Heu_{algo}, both produces the same results.



(f) Power utilization by network and computing stacks with migration for *Type#2* initial placement in $Comb_{algo}$ and Heu_{algo}, both produces the the same results.

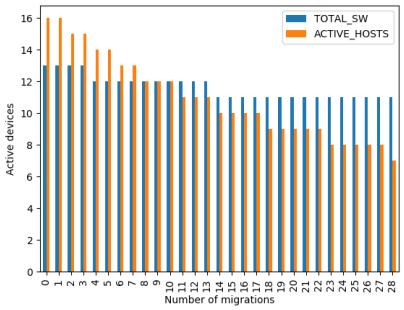
Figure 3: The change of active devices and power consumption required in $MinMig_{algo}$, $Comb_{algo}$, and Heu_{algo} for initial VM placement distribution of *Type#1* and *Type#2*. PW_SW: Power consumed by switches, PW_HOSTS: Power consumed by hosts, PW_TOTAL: Joint power consumed by network-computing stacks, TOTAL_SW: Number of active switches, ACTIVE_HOSTS: Number of active hosts.

B. Initial VM Placements

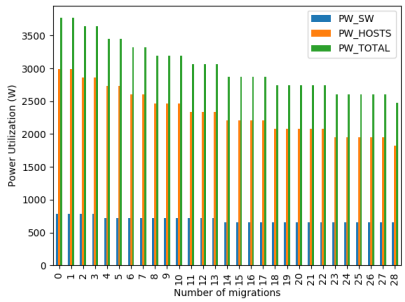
We present our results with three different initial placement distributions, namely – *Type#1*, uniformly distributed; *Type#2*, and *Type#3*, randomly distributed by constrained to between 2 and 7 VMs per host (see Table II). We consider three initial representative placements to explain the trade-offs and differences in results between the three presented algorithms discussed in the next section. In a large data center, different collections of servers may consist of nonuniform devices and switches, but in a single rack, the devices are generally homogeneous. The current manuscript limits the focus to the homogeneous form of the problem. To this end, we consider the highest feasible workload of each host to be 10 VMs.

C. Source-Destination Host Selection for Migration

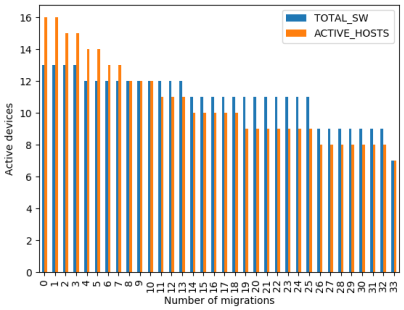
To ensure the best power consumption possible for the computational component, VMs are selected for migration from the mostly empty (but not totally empty) host of $Source\#$ list, and sent to the least empty (but not full) host $Destination\#$ list. This strategy leads to quick convergence towards a power reduction state, so we apply it in all three algorithms. Our energy calculator module measures the total



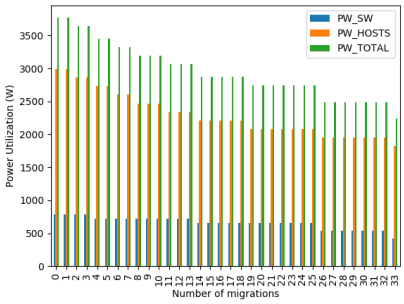
(a) Number of active devices changes with migration for *Type#3* initial placement in $MinMig_{algo}$



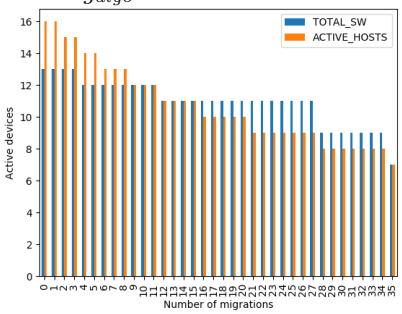
(b) Power utilization by network and computing stacks changes with migration for *Type#3* initial placement in $MinMig_{algo}$



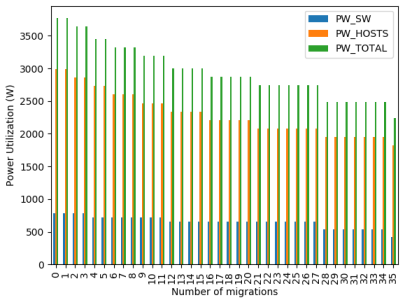
(c) Number of active devices changes with migration events for *Type#3* initial placement in $Comb_{algo}$



(d) Power utilization by network and computing stacks changes with migration for *Type#3* initial placement in $Comb_{algo}$



(e) Number of active devices changes with migration for *Type#3* initial placement in Heu_{algo}



(f) Power utilization by network and computing stacks with migration for *Type#3* initial placement in Heu_{algo}

Figure 4: The change of active devices and power consumption required in $MinMig_{algo}$, $Comb_{algo}$, and Heu_{algo} for initial VM placement distribution of *Type#3*. PW_SW: Power consumed by switches, PW_HOSTS: Power consumed by hosts, PW_TOTAL: Joint power consumed by network-computing stacks, TOTAL_SW: Number of active switches, ACTIVE_HOSTS: Number of active hosts.

power, before and after the migration event, to log the effectiveness of the migration events. Next, we compare the presented algorithms and analyze the reduction of power consumption over migration events in the DC.

D. Analysis of Power Reduction Rate

Figure 3a, and 3b illustrate the required number of devices needed at all the stages of migration, and the power consumption incurred for *Type#1* uniform VM distribution for all three algorithms described in section IV.

Further, Figure 3c-3f explains the difference of the other two algorithms ($Comb_{algo}$ & Heu_{algo}) from the baseline $MinMig_{algo}$, through *Type#2* distribution. For *Type#2* distribution, $MinMig_{algo}$ can pack the VMs with the minimum number of VM migrations but is unable to scale the power consumption compared to the other two algorithms.

Figure 3c, and 3d illustrate the required number of devices, and the power consumption incurred by *Type#2* VM distribution for the baseline algorithm $MinMig_{algo}$ at all the stages of VM migration. Figure 3e, and 3f illustrate the required number of devices, and the power consumption for $Comb_{algo}$, and Heu_{algo}, results are exactly same for both the algorithms on the *Type#2* VM distribution.

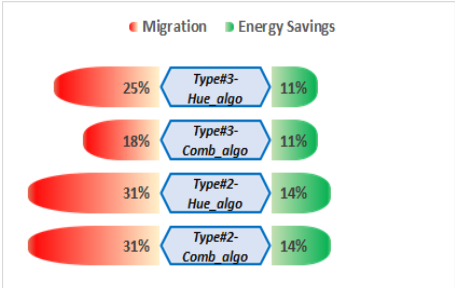
***Key Takeaways from Figure 3*:** Figure 3a-3b shows, for initial VM placement of *Type#1*, all three algorithms — (A) result in the same number of migration (33) to scale the CDC at its highest possible energy reduction state, (B) result in the same minimum amount of energy consumption at the final state, (C) result in the same energy reduction rate via all the iterations over migration events that took place.

Figure 3c-3f demonstrates, for initial VM placement of *Type#2* — (A) $MinMig_{algo}$ results in 26 migrations, where as $Comb_{algo}$ and Heu_{algo} require 34 migrations to scale the CDC to its minimum possible power consumption state, (B) $MinMig_{algo}$ consumes more power than both the $Comb_{algo}$ and Heu_{algo} ($Comb_{algo}$ and Heu_{algo} took the same amount) at the final state, (C) $MinMig_{algo}$ results in a better power reduction rate in comparison to the other two algorithms for some iterations over the migration events that took place , (D) while *Type#2* VM distribution can explain the difference between $Comb_{algo}$ & Heu_{algo} and $MinMig_{algo}$, it is unable to explain the difference between $Comb_{algo}$ and the Heu_{algo}.

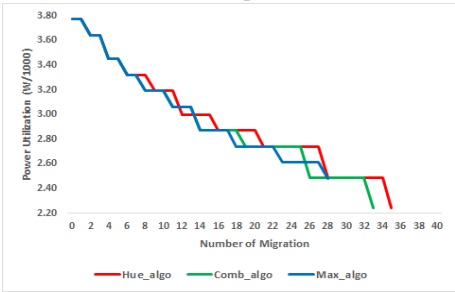
Figure 4 shows that the *Type#3* distribution can expose

Table III: VM distributions in servers and effectiveness of the algorithms (MM_{algo} is $MinMig_{algo}$)

Efficacy of algorithms When at Power Scaled State	Type#1			Type#2			Type#3		
	MM_{algo}	Heu_{algo}	$Comb_{algo}$	MM_{algo}	Heu_{algo}	$Comb_{algo}$	MM_{algo}	Heu_{algo}	$Comb_{algo}$
Number of Used Hosts	5	5	5	7	7	7	7	7	7
Power Taken By Hosts (KW)	1274	1274	1274	1794	1794	1794	1820	1820	1820
Power Taken by Switches (KW)	360	360	360	720	420	420	660	420	420
Total Power (KW)	1634	1634	1634	2514	2214	2214	2480	2240	2240
Number of Migration to Reach Power Scaled State	33	33	33	26	34	34	28	35	33



(a) Performance of Heu_{algo}, $Comb_{algo}$ in compare to $MinMig_{algo}$ for *Type#2*, and *Type#3* VM distribution presented in Table II



(b) Relative decrements of power utilization with the number of migration for *Type#3* VM distribution presented in Table II

Figure 5: Relative performance of the three presented algorithms – $MinMig_{algo}$, $Comb_{algo}$, and Heu_{algo}

differences in efficiency between all three algorithms. For *Type#3* distribution, $MinMig_{algo}$ can pack all the running VMs with the least number of VM migrations, but cannot scale down the power consumption to the same level as $Comb_{algo}$ and Heu_{algo}. Again, although both $Comb_{algo}$ and Heu_{algo} can scale down the power consumption to a similar lower state, Heu_{algo} may need a higher number of VM migrations. In contrast, $Comb_{algo}$ can perform better in terms of the number of migrations needed, by compromising the computational cost just to determine the *Destination#* list. Table III summarizes the results for all distribution Types while attaining the power scaled state.

***Key Takeaways from Figure 4*:** For initial placement *Type#3* — $MinMig_{algo}$ results in 28 migrations, where as $Comb_{algo}$ takes 33, and Heu_{algo} takes 35 migrations to scale the DC at its highest possible power reduced state. But, $MinMig_{algo}$ still consumes 11% more power than $Comb_{algo}$ and Heu_{algo} ($Comb_{algo}$ & Heu_{algo} took the same amount) at the final state. $MinMig_{algo}$ reaches its lowest point, with comparatively fewer migrations (28), while $Comb_{algo}$ takes more migrations (33), it can scale-down the energy consumption further. Finally, Heu_{algo} (35 migrations) can lead to the same lowest energy consumption state as $Comb_{algo}$, but at the cost of 2 more migrations, but with considerably less computational cost.

Relative performance: In Figure 5a, it can be observed that compared to the baseline algorithm $MinMig_{algo}$, the other two $Comb_{algo}$ and Heu_{algo} can lead to 11% more power reduction, but with the penalty of completing 18% more VM migrations for $Comb_{algo}$, and 25% more VM migrations for Heu_{algo}, for $Type\#3$ initial VM distribution. Again, $Comb_{algo}$ can perform better than Heu_{algo} in terms of the required number of migration to reach to the highest power reduced state, but the disadvantage of $Comb_{algo}$ is its exponential computation complexity.

Figure 5b compares the three presented algorithms in terms of rate of the decrease of power utilization per migration for the *Type#3* data. Until the 7th migration, all three algorithms perform similarly, then Strategy 1 ($MinMig_{algo}$) remains as the best until after the 12^{th} and 13^{th} migration, where Heu_{algo} performs better, and the 26^{th} and 27^{th} migration, where $Comb_{algo}$ performs better. Although $MinMig_{algo}$ performs better and requires fewer migrations, it reaches a fixed point after that. On the other hand, $Comb_{algo}$ can scale-down the energy state further with more migrations (33), but deciding the hosts to have remained active is computationally more complex, whereas the Heu_{algo} has the power to scale-down the energy state similar to the $Comb_{algo}$ but at the cost of 2 more migrations.

VI. Related Work

Saving energy in cloud data centers has been a fertile area of research from around the last one and a half decades, but the focal point of this research investigation remained mostly on Infrastructure (for example, maintaining cooling systems of CDC) and computing parts. The advent of the OpenFlow standard [18] that defines the communication interface between the control and forwarding planes of the SDN framework created opportunities for alternative strategies for scaling down power consumed by DC networks. Until now few initiatives have jointly optimized network

and host energy together, for example, SLA-awareness [25], network, and data awareness [24], workflow-awareness [26]. The idea of encapsulating the energy scaling-down improvement by combining both the network switch and server components while VM migration remained an important and unexplored topic. Most of the research to date has focused on energy-efficient VM allocation and management techniques [23],[21],[14],[19]. Next, we are reporting some of the most relevant work in brief.

Heller et al. [11] introduced the idea of the ElasticTree, which can adjust different network elements dynamically with the changing of traffic loads in a data-center. This approach consists of finding the minimal subset of network elements, which will be sufficient to forward information across the whole network by concatenating the traffic through selected links so that the inactive network elements can be switched off to save the energy. Paliwal and Shrimankar [21] also explored the mechanism of switching off network elements that are have low load and migrating their traffic to some other elements in a multi-path network architecture to save network energy, and improve bandwidth utilization. They simulated the network over NS-3, and captured the traffic through Wireshark. However none of this work considers the power consumption of edge switches and the computing stacks of a CDC.

Links near the core (higher layers) in a DCN are typically congestion prone due to the oversubscribed nature of usage. Cziva et al. [12], [22] explored algorithms to alleviate the congestion at the higher layers of the DC topology. Cui et al. [13] considered flows traversing different middleboxes in DCN and proposed policies for communication cost reduction through VM migration while meeting the network policy requirements. Fizi et al. [20] have proposed a dynamic load balancing algorithm for re-routing the flow and modifying the entries of the flow table to handle congestion control in SDN based DCs. Jiang et al. [29] considered the dynamically changing nature of the traffic loads of a DC and minimized the traffic cost by managing the placement of VMs and network routing. They used Markov approximation to find a near-optimal solution in a feasible time. However, their work only focused on the communication and congestion control involved in a DCN, energy consumption of computing stacks have not been considered.

Son et al. [25] proposed a dynamic overbooking strategy which jointly leveraged virtualization capabilities and SDN for VM and traffic consolidation. Their main focus was on SLA-awareness with the presence of dynamic overbooking strategies. Contrary to our work, they simulated some of the properties of SDN, rather than using a real SDN controller such as – ONOS, OpenDaylight, and Ryu. They only considered exceeding expected response time as their SLA, and did not take into account the number of VM migrations. Son and Buyya [14] proposed a priority-aware algorithm that places VMs running high-priority applications to closely connected hosts in order to reduce the chance of network congestion caused by other tenants in SDN enabled clouds. However, again they only measured QoS by response time, and did not use a real SDN controller or even emulator. Similarly, they ignored the discussion on the number of migrations.

Han et al. [23] derived an optimal solution for dynamic VM management by formulating it as a large-scale Markov decision process problem. They only focused on modeling the dynamics of CPU usage and considered computational power (CPU time) as the main factor determining the power consumption of the servers. They considered other resources like - network, disk, RAM as static without allowing oversubscription. Zheng et al. [27] focused on the correlation between VMs to find optimal VM placement considering host and network resources. Their model considered the power taken by chassis, switch, server, and each port. They proposed a VM placement and traffic consolidation algorithm based on the correlation coefficient of traffic flows. Jin et al. [28] formulated an integer linear program to manage joint host-network problems in order to optimize the power consumption of a cloud data center. Their solution focused on finding the best host for placing VM using a depth-first search to combine host and network effectively.

VII. Conclusions and Future directions

This paper identified and investigated a trade-off between the number of migrations, joint host-network power consumption, and the computational complexities of the migration algorithms, namely $MinMig_{algo}$, $Comb_{algo}$, and Heu_{algo} in respect of three different initial VM placements in a CDC. We tested our proposed algorithms by observing the power consumption improvement combining the network and computing stacks showing the progress of VM migrations in a data center. We substantiated our objectives on a proposed framework where the network stack is emulated with Mininet plugged with the ONOS network controller and the server stacks and VM virtualized characteristics are simulated through Python script using the API of Mininet. Experimental results showed that $MinMig_{algo}$ can minimize the number of VM migrations, not ensuring the lowest power consumption state. Where as, $Comb_{algo}$ can balance between reducing power consumption and the number of migrations by giving higher precision to the power reduction, but at the cost of exponential computational complexity.

On the other hand, the heuristic based algorithm Heu_{algo} was able to effectively balance between both of these bounds, with very few compromises in the number of migrations, but doing so with a viable time complexity of $\mathcal{O}(K \times \log K)$, where K is the order of DCN tree topology. These results suggest that the proposed approach may be of considerable value to operators seeking to balance conflicting needs in CDC power management.

Potential future directions of this work include but not limited to — (a) incorporating the features of live migration,

measuring the related overhead and power consumption in detail, (b) extending and evaluating this work with heterogeneous setup of cloud data center, and (c) exploring adaptations of these algorithms to account for highly time variant renewable energy power sources.

REFERENCES

[1] D. J. Schumacher and W. C. Beckman, "Data center cooling system," U.S. Patent 6 374 627, Apr. 23, 2002.

[2] E. Pakbaznia and M. Pedram. 2009. "Minimizing data center cooling and server power costs," In *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, pp. 145–150.

[3] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, and S. Poole. 2010, August. "Energy-efficient application-aware online provisioning for virtualized clouds and data centers," In *International Conference on Green Computing*, pp. 31–45.

[4] H. Goudarzi and M. Pedram. 2011, July. "Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems," In *2011 IEEE 4th International Conference on Cloud Computing*, pp. 324–331.

[5] H. Goudarzi, M. Ghasemazar, and M. Pedram. 2012, May. "SLA-based optimization of power and migration cost in cloud computing," In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 172–179.

[6] Zhang et al. 2012. "Dynamic energy-aware capacity provisioning for cloud computing environments," In *Proceedings of the 9th international conference on Autonomic computing*, pp. 145–154.

[7] T. M. Lynar, Simon, R. D. Herbert, and W. J. Chivers. "Reducing energy consumption in distributed computing through economic resource allocation," *International Journal of Grid and Utility Computing*, vol. 4, no. 4, (2013), pp. 231–241.

[8] Q. Xilong and X. Peng. "An energy-efficient virtual machine scheduler based on cpu share-reclaiming policy," *International Journal of Grid and Utility Computing*, vol. 6, no. 2, (2015), pp. 113–120.

[9] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder. 2009. "Understanding and abstracting total data center power," In *Workshop on Energy-Efficient Design*, Vol. 11.

[10] A. Andrae and T. Edler. "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, no. 1, (June, 2015), pp. 117–157.

[11] B. Heller et al. "Elastictree: Saving energy in data center networks," In *Nsdi*, vol. 10, (April, 2010), pp. 249-264.

[12] R. Cziva, S. Jouët, D. Stapleton, F. P. Tso, and D. P. Pezaros. "SDN-based virtual machine management for cloud data centers," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, (2016), pp.212-225.

[13] L. Cui, F. P. Tso, D. P. Pezaros, W. Jia and W. Zhao. "Plan: Joint policy-and network-aware vm management for cloud data centers," *IEEE Transactions on Parallel and Distributed Systems*, 28(4), (2016), pp.1163-1175.

[14] J. Son and R. Buyya. "Priority-aware VM allocation and network bandwidth provisioning in software-defined networking (SDN)-enabled clouds," *IEEE Transactions on Sustainable Computing*, 4(1), (2018), pp.17-28.

[15] D. A. Jiménez. Study Materials from Computer Science and Engineering Department at Texas A&M University. [Online]. Available at: http://hpca23.cse.tamu.edu/taco/utsa-www/ut/utsa/cs3343/lecture25.html Accessed - June, '20]

[16] Mininet. [Online: mininet.org Accessed - Aug'19]

[17] Open Network Operating System. [Online: https://wiki.onosproject.org/display/ONOS/ONOS Accessed - Aug'19]

[18] OpenFlow standard: Communication Interface between Control and Forward Plane of an SDN framework. [Online: https://en.wikipedia.org/wiki/OpenFlow Accessed - July'20]

[19] Khaoula Hamdi and Meriam Kefi. 2016, March. "Network-aware virtual machine placement in cloud data centers: An overview," In 2016 *International Conference on Industrial Informatics and Computer Systems (CIICS)* (pp. 1-6). IEEE.

[20] F. S. Fizi and S. Askar. 2016, September. "A novel load balancing algorithm for software defined network based data-centers," In 2016 *IEEE International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*, pp. 1-6.

[21] M. Paliwal and D. Shrimankar. 2019. "Effective Resource Management in SDN enabled Data Center Network based on Traffic Demand," *IEEE Access*, pp.69698-69706.

[22] R. Cziva. 2018. "Towards lightweight, low-latency network function virtualisation at the network edge," (*Doctoral dissertation*, University of Glasgow).

[23] Z. Han, H. Tan, R. Wang, G. Chen, Y. Li, and F. C. M. Lau. "Energy-Efficient Dynamic Virtual Machine Management in Data Centers," *IEEE/ACM Transactions on Networking (TON)*, 27(1), (2019), pp.344-360.

[24] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya. "An algorithm for network and data-aware placement of multi-tier applications in cloud data centers," *Journal of Network and Computer Applications*, 98, (2017), pp.65-83.

[25] J. Son, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya. "SLA-aware and energy-efficient dynamic overbooking in SDN-based cloud data centers," *IEEE Transactions on Sustainable Computing*, 2(2), (2017), pp.76-89.

[26] A. Jayanetti and R. Buyya. 2019. "J-OPT: A Joint Host and Network Optimization Algorithm for Energy-Efficient Workflow Scheduling in Cloud Data Centers," In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, IEEE/ACM, pp. 199-208.

[27] K. Zheng, X. Wang, L. Li, and X. Wang. 2014, April. "Joint power optimization of data center network and servers with correlation analysis," In *IEEE INFOCOM 2014 Conference on Computer Communications*, pp. 2598-2606.

[28] H. Jin et al. 2013, May. "Joint host-network optimization for energy-efficient data center networking," In *IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 623-634.

[29] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang. 2012, March. "Joint VM placement and routing for data center traffic engineering," In *Proceedings IEEE INFOCOM*, pp. 2876-2880.

[30] A. Beloglazov and R. Buyya. "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE transactions on parallel and distributed systems*, vol. 24, no. 7, (2012), pp. 1366-1379.

[31] A. Kertész, D. D. József, and A. Benyi. "A pliant-based virtual machine scheduling solution to improve the energy efficiency of iaas clouds." Journal of Grid Computing 14, no. 1 (2016): 41-53.