RI 10013, 19 November 2010 Computer Science

IBM Research Report

VMFlow: Leveraging VM Mobility to Reduce Network Power Costs in Data Centers

Vijay Mann
IBM Research - India
vijamann@in.ibm.com

Partha Dutta
IBM Research - India
parthdut@in.ibm.com

Shivkumar Kalyanaraman
IBM Research - India
shivkumar-k@in.ibm.com

Avinash Kumar[†]
Indian Institute of Technology, Delhi
avinash.ee@student.iitd.ac.in

IBM Research Division Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

[†]This work was done while the author was a summer intern at IBM Research - India

Abstract

Networking costs play an important role in the overall costs of a modern data center. Network power, for example, has been estimated at 10-20% of the overall data center power consumption. Techniques to save power in data centers have traditionally focussed on server power reduction through Virtual Machine (VM) migration and server consolidation. Most VM placement and migration techniques do not taken into account network topology and current network traffic. On the other hand, recent techniques to save network power have not yet utilized the various degrees of freedom that current and future data center will soon provide. These include VM migration capabilities across the entire data center network, on demand routing through programmable control planes, and high bisection bandwidth network architectures such as VL2, Portland and BCube.

This paper presents VMFlow: a framework for placement and migration of VMs that takes into account both the network topology as well as network traffic demands, to meet the objective of network power reduction while satisfying as many network demands as possible. We present network power aware VM placement and demand routing as an optimization problem. We show that the problem is NP-complete, and present a fast heuristic for the same. Next, we present the design of a simulator that implements this heuristic and simulates its runs over a data center network with a CLOS topology. Our simulation results using real data center traces demonstrate that, by applying an intelligent VM placement heuristic, VMFlow can achieve 15-20% additional savings in network power while satisfying 50-60% more network demands as compared to recently proposed techniques for saving network power.

I. Introduction

Networking costs play an important role in data center costs. These include capital expenditure (CAPEX) costs such as cost of networking equipment which has been estimated at 15% of total costs in a data center [1]. Other networking costs include operational expenditure (OPEX) such as power consumption by networking equipment. It has been estimated that network power comprise of 10-20% of the overall data center power consumption [2]. It was 3 billion kWh in the US alone in 2006.

Traditionally, data center networks have comprised of single rooted tree network topologies which are ill-suited for large data centers, as the core of the network becomes highly oversubscribed leading to contention [3]. To overcome some of these performance problems of traditional data center networks, such as poor bisection bandwidth and poor performance isolation, recent research in data center networks has proposed new network architectures such as VL2 [3], Portland [4], and BCube [5]. However, it has been contended that all data center applications may not require full bisection bandwidth and all this excessive bandwidth may lie unutilized [6]. Data centers are also increasingly adopting virtualization and comprise of physical machines with multiple Virtual Machines (VMs) provisioned on each physical machine. These VMs can be migrated at downtime or at runtime (live). Furthermore, emergence of programmable control plane in switches through standardized interfaces such as Openflow [7], has enabled on demand changes in routing paths.

With the above recent advances, opportunities have emerged to save both network CAPEX and OPEX. Network CAPEX is being reduced through several measures such as increased utilization of networking devices, and a move towards cheaper and faster data plane implemented in merchant silicon, while all the intelligence lies in a separate sophisticated control plane [7]. Components of OPEX, such as network power costs, are being reduced through techniques that switch off unutilized network devices as the data center architectures move to higher levels of redundancy in order to achieve higher bisection bandwidth. However, most of the recent techniques to save network power usage do not seem to utilize all degrees of freedom that current and future data center will soon provide. For instance, recent techniques to save network power in [2] exploit the time-varying property of network traffic and increased redundancy levels in modern network architectures, but do not consider VM migration. On the other hand, current VM placement and migration techniques such as [8] mainly target server power reduction through server consolidation and do not taken into account network topology and current network traffic. A recent work by Meng et. al [9] explored network traffic aware VM placement for various network architectures. However, their work did not focus on network power reduction.

This paper presents VMFlow: a framework for placement and migration of VMs that takes into account both the network topology as well as network traffic demands, to meet the objective of network power reduction while satisfying as many network demands as possible. We make the following contributions:

1) We formulate the VM placement and routing of traffic demands for reducing network power as an optimization problem.
2) We show that a decision version of the problem is NP-complete, and present a fast heuristic for the same.
3) We present the design of a simulator that implements this heuristic and simulates its runs over a data center network with a CLOS topology.

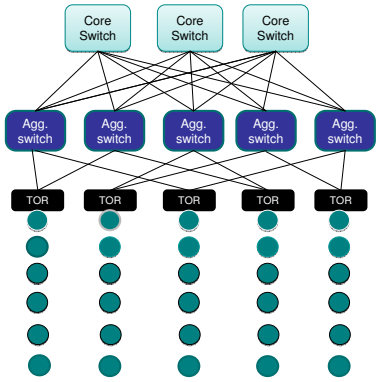

Fig. 1. CLOS network topology

4) We validate our heuristic and compare it to other known techniques through extensive simulation. Our simulation uses real data center traces and the results demonstrate that, by applying an intelligent VM placement heuristic, VMFlow can achieve 15-20% additional savings in network power while satisfying 50-60% more network demands as compared to recently proposed techniques for saving network power.

The rest of this paper is organized as follows. We give an overview of related research in Section II. Section III presents the technical problem formulation. Section IV describes our greedy heuristic. We describe the design and implementation of our simulation framework and demonstrate the efficacy of our approach through simulation experiments in Section V. Finally, we conclude the paper in Section VII.

II. Background and Related Work

Data center network architectures. Conventional data centers are typically organized in a multi-tiered network hierarchy [10]. Servers are organized in racks, where each rack has a Top-of-Rack (ToR) switch. ToRs connect to one (or two, for redundancy) aggregation switches. Aggregation switches connect to a few core switches in the top-tier. Such a hierarchical design faces multiple problems in scaling-up with number of servers, e..g, the network links in higher tiers become progressively over-subscribed with increasing number of servers. Recently, some new data center network architectures have been proposed to address these issues [3]–[5].

VL2 [3] is one such recently proposed data center network architecture (Figure 1), where the aggregation and the core switches are connected using a CLOS topology, i.e., the links between the two tiers form a complete bipartite graph. Network traffic flows are load balanced across the aggregation and core tier using Valiant Load Balancing (VLB), where the aggregation switch randomly chooses a core switch to route a given flow. VL2 architecture provides higher bisection bandwidth and more redundancy in the data center network.

Traffic-aware VM placement. VM placement has been extensively studied for resource provisioning of servers in a data center, including reducing server power usage [8]. Some recent papers have studied VM placement for optimizing network traffic in data centers [9], [11]. In [9], the authors investigate network traffic aware virtual machine placement. Given the traffic matrix in a data center, and the communication cost between every pair of servers, the paper presents algorithms for placing VMs at the servers such that the total communication cost is minimized. In [11], the authors study the coupled placement of computation and storage resources for applications in a data center. Given a function that specifies for each 3-tuple (A, C, S), the cost incurred by application A if it places the computation at node C and the storage at node S, the paper presents near optimal techniques to place the computation and storage resources of all applications such that the total cost is minimized. The above two approaches differ from VMFlow in two important ways:(a) the techniques do not optimize for network power, and (b) their solutions do not specify the routing paths for the network demands.

Network power optimization. Most of the recent research on data center energy efficiency has focused on reducing the two major components of data center power usage: servers and cooling [1], [2]. Recently, some papers have focussed on reducing the power consumed by networking elements (which we call, network power) in a data center [2], [12], [13]. In the ElasticTree approach presented in [2], a network power manager dynamically turns on or off the network elements (switches and links), and routes the flows over the active network elements, based on the current network traffic demands. In the primary technique presented in ElasticTree, called greedy bin-packing, every

demand is considered sequentially, and the demand is routed using the leftmost path that has sufficient capacity to route the demand. Here, a path is considered to be on the left of another path if, at each switch layer of a structured data center topology, the switch on the first path is either to the left or is identical to the switch on the second path. ElasticTree also proposes an topology-aware approach that improves upon the computation time as compared to greedy bin packing - however, it does not change the network power in the solution. The VMFlow framework that we present in this paper fundamentally differs from ElasticTree because we exploit the flexibility of VM placements that is available in the modern data centers. Also, for a given demand, we jointly perform the VM placement and flow routing by greedily selecting the VM placements as well as the routing paths that require minimum additional network power.

More recently, for enterprise networks, [12] describes a system for monitoring network traffic and power usage. Based on the obtained measurements, the paper proposes various techniques for moving towards an energy proportional network. In [13], the authors propose a technique for reducing the number of switches used for routing a given traffic matrix over a given data center topology. Neither of these approaches consider VM placement and migration.

III. Network-Aware VM Placement Problem (NAVP Problem)

In this section we present the Network-Aware Virtual Machine Placement (NAVP) Problem. Our problem formulation follow the virtual machine placement problem in [9], and the elastic tree problem in [2].

A. Problem Formulation

Input. The data center network is composed of network switches and links that connect the hosts (physical servers). The data center network topology is modeled using a weighted directed graph $G = (V, E)$, where V are the set of vertices and the $E \subseteq V \times V$ is the set of directed edges. An link $e = (u, v)$ has capacity (maximum supported rate) of $C(e)$. There are three types of vertices in V: the switches, the hosts, and a special vertex v_E. Vertex v_E models the external clients that communicate with the data center. The edges represent the communication links between the switches, and between the switches and the hosts. (We use edges and links interchangeably in this paper.) Let there be n hosts $H = \{h_1, \ldots, h_n\}$, and q switches $W = \{w_1, \ldots, w_q\}$.

We consider a set of m Virtual Machines (VMs) $M = \{vm_1, \ldots, vm_m\}$, where $m \leq n$, and at most one VM can be placed on a host. The network traffic source or destination is one of the m VMs or an external client. We model the traffic to and from any external clients as traffic to and from v_E, respectively. Let M' be $M \cup \{v_E\}$. We are given a set of K demands (rates) among the nodes in M', where the j^{th} demand requires a rate of r_j from source $s_j \in M'$ to destination $d_j \in M'$, and is denoted by (s_j, d_j, r_j).

When a switch w_i or a link e is powered on, let $P(w_i)$ and $P(e)$ denote the power required to operate them, respectively. A *VM placement* is a (one-to-one) mapping $\Pi : M \rightarrow H$ that specifies that the mapping of VM vm_i to host $h_{\Pi(vm_i)}$. In addition, we assume that in all VM placements, v_E is mapped to itself.

Constraints. We model the VM placement problem as a variant of the multi-commodity flow problem [14]. A *flow assignment* specifies the amount on traffic flow on every edge for every source-destination demand. We say that a flow assignment on G satisfies a set of demands if the following three constraints holds for the flow assignment: (1) *edge capacity:* the total flow assigned on each edge does not exceed the edge capacity, (2) *flow conservation:* for each demand, the total incoming flow is equal to the total outgoing flow at each node, except at the source and destination nodes of the flow, and (3) *demand satisfaction:* for each demand, the total outgoing flow at the source, and the total incoming flow at the destination is equal to the rate of the demand.

We consider another set of constraints that result from the power requirements: (1) a link can be assigned a non-zero flow only if it is powered on, and (2) a link can be powered on only if the switches that are the link's end nodes are powered on. Thus, the *total (network) power* required by a flow assignment is the sum of the power required for powering on all links with non-zero flow assignment, and the power required for powering on all switches that are end nodes of some link that has a non-zero flow assignment.

Due to adverse effect of packet reordering on TCP throughput, it is undesirable to split a traffic flow of a source-destination demand [15]. Therefore, the NAVP problem requires that a demand must be satisfied using a network flow that uses only one path in the network graph (unsplittable flow constraint).

The problem. Given the above-mentioned K demands among the nodes in M', we say that a given VM placement Π is *feasible* if there is a flow assignment on G that satisfies the following K demands among the hosts: the j^{th}

demands requires a rate of r_j from source host $\Pi(s_j)$ to destination host $\Pi(d_j)$. Then the Network-Aware VM Placement (NAVP) problem is stated as follows: among all possible feasible VM placements, find a placement and an associated flow assignment that has the minimum total power.

B. Problem Complexity

We now show that the following decision version of the NAVP problem is NP-complete: given a constant B, does there exist a feasible VM placements and an associated flow assignment that has total power less than or equal to B. We show the NP-completeness by reduction from the bin packing problem [16].

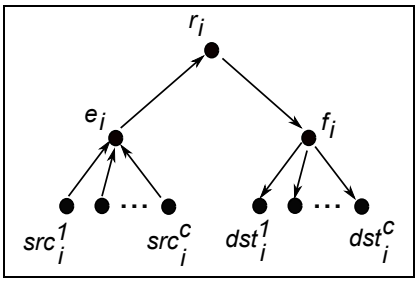

Fig. 2. Component i in the NAVP instance in the proof of Theorem 1

Theorem 1. *The decision version of the Network-Aware VM Placement (NAVP) problem is NP-complete.*

Proof: Given a VM placement and its associated flow assignment it is straightforward to verify in polynomial time whether the VM placement is feasible and whether its total power is less than B. Thus, the decision version of NAVP is in NP.

In the decision version of the bin packing problem, given c items with sizes $a_1, \ldots, a_c \in (0, 1]$ and a constant D, we need to find a packing of items into unit sized bins such that the number of used bins is at most D [16]. (Note that we need at most c bins.) From an instance of the bin packing problem, we construct an instance of the NAVP problem as follows.

- The directed graph G consists of c components, one corresponding to each bin (see Figure 2).
  - In each component i ($1 \leq i \leq c$), there is a root node r_i, and two children of the root node, e_i and f_i. Also, nodes e_i and f_i each has c children leaf nodes src_i^j and dst_i^j, respectively ($1 \leq j \leq c$). Here, the leaf nodes src_i^j and dst_i^j are the hosts and other nodes (r_i, e_i and f_i) are switches.
  - There is a directed edge from every src_i^j to e_i, and from f_i to every dst_i^j. Also, there is a directed edge from e_i to r_i, and a directed edge from r_i to f_i. All edges have capacity 1.
- The power required to operate each r_i switch is 1, and the power required to operate each e_i and f_i is 0. Also, the power required to operate a link is 0.
- There are $2c$ VMs $\{x_1, \ldots, x_{2c}\}$ and c demands, where the demand i ($1 \leq i \leq c$) is from x_i to x_{i+c} and has rate a_i.

We now show that, if the bin packing instance has a solution with D bins, then the above instance of NAVP has a solution with total power D, and vice-versa. Suppose that, there is a solution to the bin packing problem instance with D bins. Consider the set of items T_i packed in bin i. For each $a_j \in T_i$, in the NAVP problem instance, we assign the flow for the demand j in the j^{th} graph component of G, and the source and the destination of the demand at host src_i^j and dst_i^j, respectively. The flow is routed as follows: $src_i^j - e_i - r_i - f_i - dst_i^j$ Note that, as the sum of sizes of the items in T_i is less than or equal to the size of the bin (i.e., 1), the edge capacities of 1 are sufficient to support all the flows assigned to the component. Thus, we have a feasible VM placement with total power D.

For the reverse direction, consider a solution of the NAVP problem instance: a feasible VM placement and associated flow assignment with total power D. For each $1 \leq j \leq n$, if the source of demand j is placed on a host in component i, then we place item j in bin i. Note that, due to the direction of the edges, in the NAVP solution, any source VM x_i ($1 \leq i \leq c$) can only be placed at a src host, and any destination VM x_i ($c+1 \leq i \leq 2c$) can only be placed at a dst host. Also, the source and the destination of a demand should be in the same component. Therefore, all flows in a component i are routed through the edge (e_i, r_i). Since the capacity of the edge (e_i, r_i) is

1, the sum of the rates of all flows in the component is 1. It follows that the sum of the item sizes placed in bin i of our assignment is 1. Thus, we have a bin packing with D bins. ■

Note that, in the above reduction there is at most one path between any pair of hosts in G. Therefore, the reduction also holds when the unsplittable flow constraint is removed from the NAVP problem (i.e., a flow satisfying a demand may take multiple paths). We call this problem the Splittable NAVP (SNAVP) problem.

IV. Heuristic Design

1: **Input:** described in Section III-A
2: **function** Initialization
3: $W_{on} \leftarrow \emptyset$; $E_{on} \leftarrow \emptyset$ {*set of switches and edges currently powered on*}
4: $H_{free} \leftarrow H$ {*set of hosts currently not occupied by a VM*}
5: $\forall e \in E$, $RC(e) \leftarrow C(e)$ {*current residual capacity of edge* e}
6: **function** VMFlow
7: select a demand (s_j, d_j, r_j) from the set of demands in the descending order of their rates
8: **if** $\Pi(s_j) = \perp$ **then** $\Phi(s_j) \leftarrow H_{free}$ **else** $\Phi(s_j) \leftarrow \{\Pi(s_j)\}$
9: **if** $\Pi(d_j) = \perp$ **then** $\Phi(d_j) \leftarrow H_{free}$ **else** $\Phi(d_j) \leftarrow \{\Pi(d_j)\}$
10: $V_{res} \leftarrow W \cup \Phi(s_j) \cup \Phi(d_j)$ {*residual nodes*}
11: $E_{res} \leftarrow \{(u, v) \in E : (RC(e) \geq r_j) \wedge (u, v \in V_{res})\}$ {*residual edges*}
12: $G_{res} \leftarrow (V_{res}, E_{res})$ {*residual graph*}
13: $\forall v \in W$, **if** $v \in W_{on}$ **then** $WT_{res}(v) \leftarrow 0$ **else** $WT_{res}(v) \leftarrow P(v)$ {*switch weights in residual graph*}
14: $\forall v \in \Phi(s_j) \cup \Phi(d_j)$, $WT_{res}(v) \leftarrow \infty$ {*host weights in residual graph*}
15: $\forall e \in E_{res}$, **if** $e \in E_{on}$ **then** $WT_{res}(e) \leftarrow 0$ **else** $WT_{res}(e) \leftarrow P(e)$ {*edge weight in residual graph*}
16: $minWt \leftarrow Min\{pathWt(G_{res}, u, v) : (u \in \Phi(s_j)) \wedge (v \in \Phi(d_j)) \wedge (u \neq v)\}$ {*described in Section IV*}
17: **if** $(minWt < \infty)$ **then** {*found a routing for this demand*}
18: $(\Pi(s_j), \Pi(d_j)) \leftarrow$ any (u, v) s.t. $(u \in \Phi(s_j)) \wedge (v \in \Phi(d_j)) \wedge (u \neq v) \wedge (pathWt(G_{res}, u, v) = minWt)$
19: assign the minimum weight path P from $\Pi(s_j)$ to $\Pi(d_j)$ to the flow for (s_j, d_j, r_j)
20: for all switches v on path P, $W_{on} \leftarrow W_{on} \cup \{v\}$
21: for all edges e on path P, $E_{on} \leftarrow E_{on} \cup \{e\}$; $RC(e) \leftarrow RC(e) - r_j$
22: $H_{free} \leftarrow H_{free} \setminus \{\Pi(s_j), \Pi(d_j)\}$
23: **else**
24: skip demand (s_j, d_j, r_j) {*cannot find a VM placement for this demand*}

Fig. 3. A greedy algorithm for NAVP

We now present a greedy heuristic for the NAVP problem. The algorithm considers the demands one by one, and for each demand, the algorithm greedily chooses a VM placement and a flow assignment (on a path) that needs minimum increase in the total power of the network. We now describe the algorithm in more details. (The pseudocode is presented in Figure 3.)

Primary variables. The algorithm maintains four primary variables: W_{on} and E_{on} which are the set of switches and edges that have been already powered on, respectively, the set H_{free} of hosts that have not yet been occupied by a VM, and a function RC that gives the residual or free capacity of the edges. Initially, none of the switches and edges are powered on, all hosts are unoccupied, and the full capacity $C(e)$ of an edge is free. For ease of presentation, for each VM v, we assume that initially $\Pi(v)$ is set to $\perp$.

In each iteration of the main loop (in function VMFlow), the algorithm selects a demand in the descending order of the demand rates, and tries to find a feasible VM placement. If a feasible VM placement is found, then the primary variables are updated accordingly; otherwise, the demand is skipped.

Residual graph. To find a feasible VM placement for a demand (s_j, d_j, r_j), the algorithm construct a residual graph G_{res}. The residual graph contains all hosts where VMs s_j and d_j can be possibly placed, and all switches. $\Phi(s_j)$ is the set of host where s_j can possibly be placed. If s_j has already been placed at a host while considering a previous demand, $\Phi(s_j)$ only contains the host $\Pi(s_j)$. Otherwise, $\Phi(s_j)$ is the set of all unoccupied hosts, H_{free}. $\Phi(d_j)$ is computed similarly. G_{res} also contains every edge among these nodes that have at least r_j residual capacity. Therefore, in G_{res}, any path between two hosts have enough capacity to route the demand.

The algorithm next focusses on finding VM placements for s_j and d_j such that there is path between the VMs that requires minimum additional power. To that end, the nodes and edges in the residual graph are assigned weights equal to the amount of additional power required to power them on. Thus, the weight of a switch v is set to 0 if it already powered on, and set to $P(v)$, otherwise. Edges are assigned weights in a similar manner. However, the weights of the hosts are set to ∞ so that they cannot be used as an intermediate node in a routing path. Next, the

weight of a path in the residual graph ($pathWt$) is defined as the sum of the weights of all intermediate edges and nodes on the path (and excludes the weights of the end nodes of the path).

VM placement. In the residual graph G_{res}, the algorithm considers all possible pairs of hosts for placing s_j and d_j. (The first element in the pair should be from $\Phi(s_j)$ and the second element from $\Phi(d_j)$.) Among all such pairs of hosts, the algorithm selects a pair of hosts that has a minimum weight path. (A minimum weight path between the hosts is found using a variant of Dijkstra's shortest path algorithm whose description is omitted here.) The source and the destination VMs for the demand (s_j and d_j) are placed at the selected hosts, and the flow assignment is done along a minimum weight path. Before considering the next demand, the algorithm updates the primary variables to reflect the VM placements and flow assignment for the current demand. Note that, sometimes G_{res} may be disconnected and each hosts may lie in a distinct component of G_{res}. (For instance, this scenario can occur when the residual edge capacities are such that no path can carry a flow of rate r_j.) In this case, there is no path between any pair of hosts, and the algorithm is unable to find a feasible VM placement for this demand. Depending on the data center service level objectives, the algorithm can either abort the placement algorithm, or continue by considering subsequent demands.

A practical simplification. We now present a simple observation to reduce the computation time of our heuristic. We observe that in most conventional and modern Data Center network architectures, multiple host are placed under a Top-of-the-Rack (ToR) switch. Thus, for a given demand, if a source (or destination) VM is placed on a host, then the parent ToR of the host needs to be turned on (if the ToR is not already on) to route the demand. Therefore, for VM placement, we first try to place both the source and destination VMs under the same ToR. If such a placement is possible, then only the common parent ToR needs to be turned on to route the demand. If no such ToR is available, then we compute a minimum weight path between all possible pairs of ToRs (where path weights include the end ToR weights), instead of computing minimum weight paths between all possible pairs of hosts, and then select the path with the minimum cost. This simplification is possible because the weight of a minimum weight path between two hosts (with different parent ToRs) is equal to the weight of a minimum weight path between their parent ToRs. The simplification significantly reduces the computation time because the number of ToRs is much lower than the number of hosts.

V. Experimental Evaluation

We developed a simulator to evaluate the effectiveness of VMFlow algorithm. It simulates a network topology with VMs, switches and links as a discrete event simulation. Each server (also called host) is assumed to host at most one VM. VMs run applications that generate network traffic to other VMs, and VMs can migrate from one node to the other. Switches have predefined power curves − most of the power is static power (i.e. power used to turn on the switch). At each time step, network traffic generated by VMs is routed through the switches to other VMs based on VM placement, network topology and available link capacities. At each time step of the simulation, we compute the total power consumed by the switches and the fraction of the total number of network demands that can be satisfied.

A. Network Topology

The simulator creates a network based on the given topology. Our network topology consisted of 3 layers of switches: the top-of-the-rack (ToR) switches which connect to a layer of aggregate switches which in turn connect to the core switches. Each ToR has 20 servers connected to it. At most one virtual machine (VM) can be mapped to a server. We assume a total of 1000 servers (and VMs). There are 50 ToR switches, and each server connects to a ToR over a 1 GBPS link. Each ToR connects to two aggregate switches over 1 or 10 GBPS links. We assume a CLOS topology between the aggregate and core switches similar to VL2 [3] with 1 or 10 GBPS links as shown in Figure 1. All switches are assumed to have a static power of 100 watts. This is because current networking devices are not energy proportional and even completely idle switches (0% utilization) consume 70-80% of their fully loaded power (100% utilization) [2].

Our simulator has a topology generator component that generates the required links between servers and ToRs, ToRs and aggregate switches, and aggregate switches and core switches. It takes in as input the number of servers and the static power values for each kind of switch. It then generates the number of ToR switches (assuming 20 servers under 1 ToR) and the number of aggregate and core switches using the formulae given in [3] for a CLOS

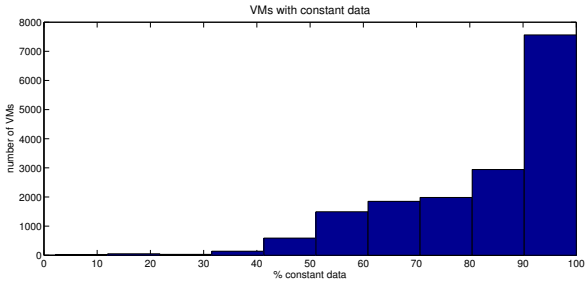

Fig. 4. Histogram of number of VMs with their percentage constant data

topology. For *(d*d)/4* ToRs, the number of aggregate switches are *d* and the number of core switches are *d/2*. For 1000 servers and 50 ToRs, this results in around 15 aggregate switches and around 8 core switches.

B. Input Data

To drive the simulator, we needed a traffic matrix (i.e. which VM sends how much data to which VM). Such fine grained data is typically hard to obtain from real data centers [17], [18] because of the required server level instrumentation. We obtained data from a real data center with 17,000 VMs with aggregate incoming and outgoing network traffic from each VM. The data had snapshots of aggregate network traffic over 15 minute intervals spread over a duration of 5 days. To understand the variance in the data, we compared the discrete time differential (Δ) with the standard deviation (σ) of the data. All data that satisfied the following criteria was considered constant:

$$-\sigma/2 \leq \Delta \leq +\sigma/2. \tag{1}$$

A histogram of percentage of data that was constant for all VMs is given in Figure 4.

It can be observed that most of the VMs have a very large percentage of data that is constant and does not show much variance. A very large variance can be bad for VMFlow, as it will mean too frequent WM migrations, whereas a low variance means that migration frequency can be kept low.

Our first task was to calculate a traffic matrix out of this aggregate per VM data. Given the huge size data, we chose data for a single day. Various methods have been proposed in literature for calculating traffic matrices. We used the simple gravity model [19] to calculate the traffic matrices for all 17,000 VMs at each timestamp on the given day. Simple gravity model uses the following equation to calculate the network traffic from one VM to another.

$$D_{ij} = \frac{D_i^{out} * D_j^{in}}{\Sigma_k D_k^{in}} \tag{2}$$

where D_i^{out} is the total outgoing traffic at VM *i*, and D_j^{in} is the total incoming traffic at VM *j*. Although, existing literature [17] points out that traffic matrices generated by simple gravity model tend to be too dense and those generated by sparsity maximization algorithms tend to be too sparse than real data center traffic matrices, simple gravity model is still widely used in literature to generate traffic matrices for data centers [9].

After generating the traffic matrices for each timestamp for the entire data center (17,000 VMs), we used the traffic matrices for the first 1000 VMs in order to reduce the simulation time required. Since gravity model tends to distribute all the traffic data observed over all the VMs in some proportion, it resulted in a large number of very low network demands. In order to make these demands significant, we used a scale-up factor of 50 for all the data.

C. Simulation Results

The simulator compares VMFlow approach with the ElasticTree's greedy bin-packing approach proposed by Heller et. al [2] to save network power. Recall that the ElasticTree's bin-packing approach chooses the leftmost path in a given layer that satisfies the network demand out of all the possible paths in a deterministic left-to-right order. These paths are then used to calculate the total network power at that time instance. For placing VMs for the ElasticTree approach, we followed a strategy that placed the VMs on nodes that had the same id as their VM id.

We assume that all the network power comprise of power for turning on the switches, and the power required for turning on each network link (i.e., for ports on the end switches of the link) is zero. We calculated the total power

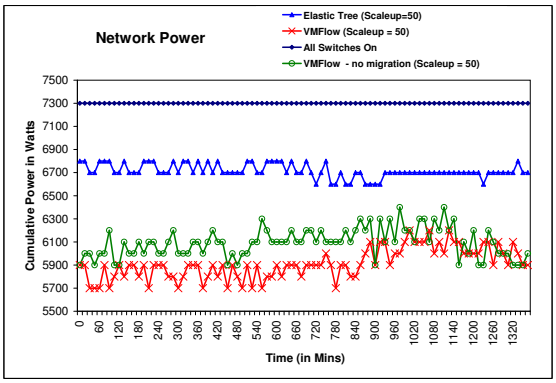

Fig. 5. Comparison of network power at different timesteps in the simulation

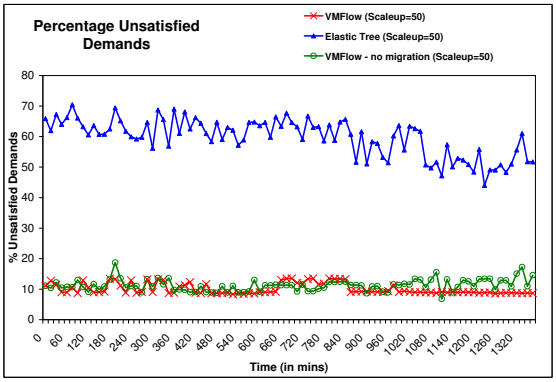

Fig. 6. Comparison of unsatisfied network demands at different timesteps in the simulation

consumed by the network and the proportion of network demands that were unsatisfied at a given timestamp for both VMFlow and ElasticTree approaches. In the first set of experiments, we simulated an oversubscribed network where the ToR-aggregate switch links and aggregate-core switch links had 1 GBPS capacity. This resulted in a 10:1 oversubscription ratio in the ToR-aggreate switch link layer. The results for the oversubscribed network are shown in Figure 5 and Figure 6, respectively.

VMFlow outperforms the ElasticTree approach by a factor of around 15% and the baseline (i.e., all switches on) by around 20% in terms of network power at any given time instance. More importantly, one can see the effectiveness of VMFlow in the percentage of network demands that remain unsatisfied. VMFlow saves all the network power while satisfying 50-60% more network demands as compared to ElasticTree approach.

Since the input data had very little variation over time, we conducted an experiment to compare VMFlow with and without any migration after the first placement was done using the VMFlow algorithm. Figure 5 and Figure 6 show the performance of VMFlow approach without any migration. One can note that even with low variance input data, VMFlow with migration outperforms VMFlow without migration slightly by a margin of 5% in terms of power. Both the approaches perform roughly the same with respect to percentage of unsatisfied network demands. This indicates that migration frequency has to be properly tuned keeping in mind the variance in network traffic. Each VM migration has a cost associated with it that depends on the size of VM, its current load and the Service Level Agreement (SLA) associated with it. We are currently working on a placement framework that will take into account this cost and the potential benefit a migration can achieve in terms of network power and network demand satisfaction.

We also compared the effect of using various scale-up factors on the input network traffic data. In this set of experiments we used a single timestamp and simulated both an oversubscribed network (ToR-aggregate and aggregate-core switch links have 1 GBPS capacity) and a network with no oversubscription(ToR-aggregate and aggregate-core switch links have 10 GBPS capacity). The results are shown in Figure 7 and Figure 8. VMFlow outperforms the ElasticTree approach consistently, both in terms of network power reduction as well as percentage unsatisfied demands in the oversubscribed network (1G). However, in the no oversubscription case (10G) the performance difference between VMFlow and ElasticTree is relatively lower. This is along expected lines, since VMFlow is expected to outperform mainly in cases where the top network layers are oversubscribed.

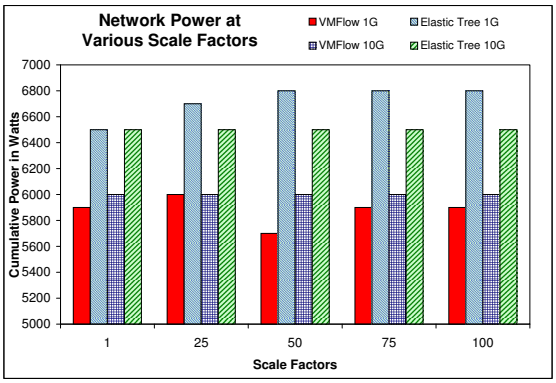

Fig. 7. Comparison of network power at various scale factors in the simulation

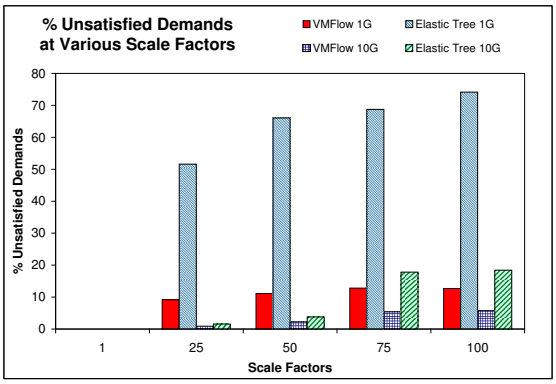

Fig. 8. Comparison of unsatisfied network demands at various scale factors in the simulation

VI. Handling Server Consolidation

In this paper, we have assumed that at most one VM can be mapped to a host; i.e., the VM placement mapping Π is one-to-one. It is, however, easy to modify our algorithm to handle the case when multiple VM are mapped to a host, i.e., in case of server consolidation. In modern data center, server consolidation is often employed to save both on the capital expenditure (e.g., data center with smaller number of hosts), and operational expenditure (e..g, any unused host can be turned off to save power) [8]. Although the basic idea of server consolidation is simple, deciding which VMs are co-located on a host is a complex exercise that depends on various factors such as fault and performance isolations, and application SLA. Thus, a group of VMs co-located during server consolidation ideally should not be migrated to different server during network power optimization. Nevertheless, a group of VMs mapped to a single host can be migrated together to a different host, provided the new host has enough resources for that group of VMs. We now describe, how our greedy algorithm can be easily extended to handle server consolidation.

We assume that we have an initial server consolidation phase which maps zero, one or more VMs to each host. A set of VMs that is mapped to the same host during server consolidation is call a VMset. We make the following two assumptions: (1) while placing VMs on the host, the only resource constraint we need to satisfy is on the compute resource, and (2) a group of VMs consolidated on a host is not separated (i.e., migrated to different hosts) during the network power optimization. Now, our greedy algorithm need two simple modification to handle server consolidation. First, instead of mapping a VM to a host, the algorithm maps VMset to a host, and the traffic demand between two VMsets is the sum of the demands between the nodes of the two VMsets. Second, in each iteration for placing the source and destination VMset of a traffic demand, the set of possible hosts ($\Pi(s_j)$ and $\Pi(d_j)$) is defined as the set of free hosts that have equal or more compute resources than the respective VMsets. We omit details of these simple modifications from this paper.

VII. Conclusion

This paper presented VMFlow, a framework for reducing power used by network elements in data centers. VMFlow uses the flexibility provided by VM placement and programmable flow-based routing, that are available in modern data centers, to reduce network power while satisfying a large fraction of the network traffic demands.

We formulated the Network-Aware VM Placement as an optimization problem, proved that it is NP-Complete, and proposed a greedy heuristic for the problem. Our technique showed reasonable improvement (15-20%) in network power and significant improvement (50-60%) in the fraction of satisfied demands as compared to previous network power optimization techniques.

As mentioned earlier, each VM migration has a cost associated with it. Similarly, the benefits that can be achieved by a VM migration in terms of network power reduction and meeting more network demands, depends on the variance in network traffic over time. In future, we plan to work on a placement framework that will take into account this cost and the potential benefit a migration can achieve in terms of network power and network demand satisfaction. We also plan to apply our technique to other network topologies and evaluate its benefits. Eventually, the placement framework needs to integrate with existing approaches for server consolidation that employ various VM packing algorithms.

REFERENCES

[1] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," in *ACM SIGCOMM CCR*, January 2009.

[2] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," in *USENIX NSDI*, April 2010.

[3] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *ACM SIGCOMM*, August 2009.

[4] R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *ACM SIGCOMM*, August 2009.

[5] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *ACM SIGCOMM*, August 2009.

[6] S. Kandula, J. Padhye, and V. Bahl, "Flyways to De-Congest Data Center Networks," in *ACM HotNets*, 2009.

[7] "The OpenFlow Switch." [Online]. Available: http://www.openflowswitch.org

[8] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *ACM/IFIP/USENIX Middleware*, 2008.

[9] X. Meng, V. Pappas, and L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," in *IEEE INFOCOM*, 2010.

[10] "Cisco: Data center: Load balancing data center services, 2004."

[11] M. R. Korupolu, A. Singh, and B. Bamba, "Coupled placement in modern data centers," in *IEEE IPDPS*, 2009.

[12] P. Mahadevan, S. Banerjee, and P. Sharma, "Energy proportionality of an enterprise network," in *1st ACM SIGCOMM Workshop on Green Networking*, 2010.

[13] Y. Shang, D. Li, and M. Xu, "Energy-aware routing in data center network," in *1st ACM SIGCOMM Workshop on Green Networking*, 2010.

[14] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows - Theory, Algorithms and Applications*. Prentice-Hall, 1993.

[15] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *Computer Communication Review*, vol. 37, no. 2, 2007.

[16] V. Vazirani, *Approximation Algorithms*. Addison-Wesley, 2001.

[17] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements and Analysis," in *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2009.

[18] T. Benson, A. A. A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," in *ACM SIGCOMM WREN Workshop*, 2009.

[19] Y. Zhang, M. Roughan, N. Duffield, and A. Greeberg, "Fast accurate computation of large-scale IP traffic matrices from link loads," in *ACM SIGMETRICS*, 2003.