Multi-resource Energy-efficient Routing in Cloud Data Centers with Network-as-a-Service

Lin Wang*§, Antonio Fernández Anta†, Fa Zhang‡, Jie Wu¶ Zhiyong Liu*‖
*Beijing Key Laboratory of Mobile Computing and Pervasive Device, ICT, CAS, China
†IMDEA Networks Institute, Spain
‡Key Lab of Intelligent Information Processing, ICT, CAS, China
§University of Chinese Academy of Sciences, China
¶Department of Computer and Information Sciences, Temple University, USA
‖State Key Laboratory for Computer Architecture, ICT, CAS, China

***Abstract*—With the rapid development of software defined networking and network function virtualization, researchers have proposed a new cloud networking model called Network-as-a-Service (NaaS) which enables both in-network packet processing and application-specific network control. In this paper, we revisit the problem of achieving network energy efficiency in data centers and identify some new optimization challenges under the NaaS model. Particularly, we extend the energy-efficient routing optimization from single-resource to multi-resource settings. We characterize the problem through a detailed model and provide a formal problem definition. Due to the high complexity of direct solutions, we propose a greedy routing scheme to approximate the optimum, where flows are selected progressively to exhaust residual capacities of active nodes, and routing paths are assigned based on the distributions of both node residual capacities and flow demands. By leveraging the structural regularity of data center networks, we also provide a fast topology-aware heuristic method based on hierarchically solving a series of vector bin packing instances. Extensive simulations show that the proposed routing scheme can achieve significant gain on energy savings and the topology-aware heuristic can produce comparably good results while reducing the computation time to a large extent.**

I. Introduction

As inter-node communication bandwidth is the principal bottleneck in data centers, there has been a large body of work on optimizing the performance of DCNs (e.g., [1], [2]). However, in order to apply these proposals to production DCNs, a lot of effort has to be undertaken, including both hardware- and software-end modifications. This is due to the specific deployment settings designed for the routing and forwarding protocols used in current data centers. As a result, incremental implementations are usually not achievable and significant effort has to be made for every single design.

The situation has been changed with the evolution of network architecture. On the one hand, researchers proposed to decouple control plane from data plane, which enables rapid innovation in network control. This idea then naturally led to the advent of Software Defined Networking (SDN). Instead of having all network nodes to run the routing protocol in a distributed manner, SDN abstracts the network control functionality to a logically centralized controller. The routing decisions are then made by the controller and pushed to network nodes through a well-defined Application Programming Interface (API) such as OpenFlow [3]. As a result, network policies can be totally implemented in the controller, which needs only very basic software modification in the event of network changes. On the other hand, the innovation in data plane has also been sped up by the technology called Network Function Virtualization (NFV) where packets are handled by software-based entities on general-purpose servers with network functions virtualized.

Taking advantage of the advancement of both control plane and data plane in networks, a new cloud networking model called *Network-as-a-Service* (NaaS) has been recently proposed [4], [5]. In the NaaS model, packet forwarding decisions are implemented based on specific application needs. Moreover, NaaS enables the design of in-network packet modification and thus in-network services, such as data aggregation, stream processing or caching can be specified by upper-layer applications. Based on this new networking model, several working examples have been studied, including in-network aggregation for big data applications [6].

However, NaaS brings new challenges to traditional network optimization problems by allowing in-network packet operations, making existing solutions inefficient or even not applicable to these problems any more. Particularly, we revisit the problem of achieving network energy efficiency in data centers and identify some new challenges under the NaaS model. With packets being processed by virtual machines running on general-purpose servers, energy-related issues become more prominent. The energy saving problem in DCNs has been widely studied and most proposals are based on the general approach of consolidating network flows and turning off unused network elements (e.g., [7], [8], [9]). In packet forwarding networks, link utilization is the most important criterion for flow consolidation. However, this is no longer valid under the NaaS model, where a network node can be congested not only by data communications, but also by the overloading of other resources such as processing units or memory. Without considering other resources, a link utilization oriented consolidation of flows may lead to severe resource

This work has been funded by the Regional Government of Madrid (CM) under project Cloud4BigData (S2013/ICE-2894) cofunded by FSE & FEDER and the National Natural Science Foundation of China (NSFC) Project for Innovation Groups 61221062.

saturation at some network nodes and to serious network instability. Therefore, it is essential to take into account multiple resources when making routing decisions under the NaaS model. To the best of our knowledge, this is the first research attempt towards multi-resource traffic engineering.

The main contributions of this paper are as follows: i) we identify new research challenges for conventional optimization problems under the NaaS model, and characterize the network energy saving problem through a detailed model with complexity analyzed; ii) we propose a greedy routing scheme where path selection is done based on the distributions of node residual capacities and flow demands; iii) by utilizing the structural property of DCNs, we provide a topology-aware heuristic which can accelerate problem solving while producing comparably good results; iv) we validate the efficiency of the proposed algorithms by extensive simulations and show that significant energy efficiency gain in NaaS-enabled DCNs can be achieved by the techniques proposed in this paper. Due to the space limit, we leave the proofs and a comprehensive discussion about the implication of the model and the proposed algorithms in the full version of this paper [10].

II. Problem Statement

While the single-resource network energy optimization problem has been well-studied [11], very little attention has been received by the energy-efficient routing problem in networks with multiple resources. With an emerging trend of software packet processing in networks, this problem has raised its significance. In the following, we provide a formal modeling of the problem and examine its complexity.

A. Preliminary Notations

We abstract a given software packet-processing network as graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the set of N nodes, each of which represents a general-purpose server with software packet processing functionalities, and $\mathcal{E}$ is the set of undirected edges representing the network links. Each node $v \in \mathcal{V}$ has limited amounts of K different types of hardware resources, namely CPU, memory, and network bandwidth, to name a few. The total amount of type-k resource is constrained by a positive capacity $C_{v,k}$ ($k \in \{1, 2, ..., K\}$). Due to the fact that packet-processing networks are usually constructed using commodity general-purpose servers, it is reasonable to assume that all the nodes in $\mathcal{V}$ are identical. Thus, for all $v \in \mathcal{V}$, we assume $C_{v,k} = C_k$ for all $k \in \{1, 2, ..., K\}$.

We define a *flow* as a sequence of data packets that possess the same entities in the packet headers such as the same source and destination IP addresses. Suppose we are given a set of M flow demands $\mathcal{D} = \{\mathsf{d}_1, \mathsf{d}_2, ..., \mathsf{d}_M\}$. The packets from the same flow d_m will be routed following a single path in order to avoid packet reordering at the destination. For all the packets from a given flow, a processing procedure is defined on every node on the flow's routing path, which is used to carry out some per-flow computation to the payloads of packets, e.g., intercepting packets on-path to implement opportunistic caching strategies [5]. Due to the fact that the data carried by the packets from the same flow generally possess the same structure (e.g., same packet size), we assume that (nearly) the same amount of computation will be applied to the packets from the same flow. As a result, we have to keep (almost) the same reservation across each type of resource on every node on the path for each flow. Each flow d_m is represented by a three-tuple $(v_m^{\mathrm{s}}, v_m^{\mathrm{t}}, \vec{R}_m)$ where v_m^{s} and v_m^{t} are the source and destination respectively, while $\vec{R}_m$ is a K-dimensional vector $(r_{m,1}, ..., r_{m,K})$ describing the amounts of resources in all types required (and reserved) for a node to process the packets from flow d_m. These resource demands can be obtained by applying the same technique used in [12]. For the sake of simplicity and without loss of generality, we assume that the $r_{m,k}$ for $m \in \{1, 2, ..., M\}$ are *normalized* by C_k for any $k \in \{1, 2, ..., K\}$, i.e. $\vec{R}_m \in [0, 1]^K$.

To quantify the performance of approximations, we term γ as the *performance ratio* of an algorithm for a minimization problem if the objective values in the solutions provided by the algorithm are upper-bounded by γ times the optimal.

B. Problem Formulation

Using the introduced notation, the energy-efficient multi-resource routing problem can be formally defined as follows. For a vector $\vec{x}$, we denote by $||\vec{x}||_\infty$ the standard ℓ_∞ norm.

Definition 1 (ENERGY-EFFICIENT MULTI-RESOURCE ROUTING (EEMR)). *Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a set of M flows $\mathsf{d}_1, ..., \mathsf{d}_M$ whose demands are characterized by $\vec{R}_1, ..., \vec{R}_M$ from $[0, 1]^K$, find a path $\mathcal{P}_m$ from v_m^{s} to v_m^{t} for each flow d_m such that $||\vec{A}_v||_\infty \leq 1$ for $v \in \mathcal{V}$ where $\vec{A}_v = \sum_{m:v \in \mathcal{P}_m} \vec{R}_m$ is the aggregation of the resource requirement vectors of flows that are routed through node v. The objective is to minimize $|\mathcal{Q}|$ where $\mathcal{Q} = \{v \mid v \in \mathcal{V} \wedge \vec{A}_i \neq (0, ..., 0)\}$ is the set of nodes that are used to carry flows.*

The EEMR problem can be formulated as a Mixed Integer Program (MIP) in the following way. We introduce two binary variables $x_{m,v}$ and y_v. The binary variable $x_{m,v}$ indicates whether flow d_m is routed through node v and y_v indicates whether node v is active or not. As the static power consumption of a node is dominant, we only consider using power-down based strategy as the main energy saving mechanism. Our objective is to minimize the number of active nodes. Note that we have an implicit assumption that feasible solutions are always achievable, that is, the network with the designed capability is able to handle the given traffic demands.

$$(\mathbb{P}_1) \quad \text{minimize} \quad \sum_{v \in \mathcal{V}} y_v$$

subject to

$$\Big|\Big| \sum_{m \in \{1,2,...,M\}} \vec{R}_m \cdot x_{m,v} \Big|\Big|_\infty \leq 1 \qquad v \in \mathcal{V}$$

$$x_{m,v} \leq y_v \qquad v \in \mathcal{V}, 1 \leq m \leq M$$

$$x_{m,v}, y_v \in \{0, 1\} \qquad v \in \mathcal{V}, 1 \leq m \leq M$$

$$x_{m,v}: \text{ flow conservation}$$

The constraints of program $\mathbb{P}_1$ are as follows: the first constraint states that the flows routed through the same node do not exceed the node resource dimensions; the second constraint tells whether a node is active or not; the third constraint ensures that each flow can only follow a single path. Flow conservation on $x_{m,v}$ forces that the nodes that flow demand d_m is routed through form a path between v_m^{s} and v_m^{t} in the network.

Note that when $K = 1$, $\mathbb{P}_1$ corresponds to the general *capacitated network design* problem which has been widely studied. For the uniform link capacitated version of the problem, Andrews, Antonakopoulos and Zhang [13] provided a polylogarithmic approximation when the capacity on each link is allowed to be exceeded by a polylogarithmic factor. Recently, [14] explored the multicommodity node-capacitated network design problem and provided a $\mathcal{O}(\log^5 n)$-approximation with $\mathcal{O}(\log^{12} n)$ congestion. However, none of the studies provide high-quality approximations in scenarios with inviolable capacity constraints. This is mainly because with strict capacity constraints, finding out whether there is a feasible solution for the problem is already NP-hard.

C. Complexity Analysis

In contrast with the traditional energy-efficient routing (i.e., capacitated network design) problem, EEMR extends the concept of "load" from single-dimensional to multidimensional, which makes the problem even computationally harder. In general, we have the following complexity results.

Theorem 1. *Solving the EEMR problem is NP-hard.*

Proof. Please refer to [10]. □

Theorem 2. *There is no asymptotic PTAS for the EEMR problem unless P=NP.*

This is directly applied from the fact that the Vector Bin Packing (VBP) problem with $K \geq 2$ is know to be APX-hard which implies that there is no asymptotic PTAS for it [15]. From the above reduction we know that actually VBP is a special case for EEMR, meaning that solving EEMR has at least the same time complexity as VBP.

III. Energy-efficient Multi-resource Routing

The complexity analysis results show that the EEMR problem is NP-hard, for which no existing exact solutions can scale to the size of current data center networks. Therefore, we resort to an intuitive approach that can provide suboptimal solutions very quickly. We detail our design in this section.

A. Key Observations

We propose a greedy routing scheme to solve the energy-efficient multi-resource routing problem. The basic principle is to use as few nodes as possible to carry all the traffic flows while satisfying the capacity constraints in all resource dimensions. More specifically, our design is based on the following two observations: i) flows preferably follow paths that consists of more *active* nodes (that already carry some traffic) as this will introduce less extra energy consumption to the network; ii) it is important to allocate routes for flows on the active nodes such that all dimensions of the resources in every active node can be fully utilized. The second observation is a new concern steaming from the multi-resource context. In the single-resource case, the only criterion for the efficiency of a node is its resource utilization, i.e., the carried traffic divided by the total capacity. As a result, steering flows to those nodes with low utilizations will lead to an energy-efficient routing solution. However, this approach is not applicable to the multi-resource case. With multiple dimensions of resources, it is not clear how to define the resource utilization of a node, thus we will not be able to make routing decisions based on node utilizations.

B. The Routing Scheme

The pseudocode of the routing scheme is shown in Algorithm 1. The algorithm runs iteratively. In each iteration, it first tries to use only the set of active nodes. By searching the flow demand list, it tries to find out a candidate flow to route on the subnetwork $\mathcal{G}_a$ composed by the active nodes and the corresponding network links connecting these nodes. Note that it is necessary to remove the nodes that are not capable of carrying the flow, that is, when the flow is carried by the nodes, at least one dimension of the resource capacities of the nodes will be violated, leading to node congestion. We denote by $\mathcal{G}_c^m$ the residual network after removing the incapable nodes and the links attached to these nodes. We then carry out function `IsConn`, a depth-first search procedure, to verify if the source and the destination of the current flow are connected in $\mathcal{G}_c^m$. If a candidate flow d_c that can be routed on $\mathcal{G}_c^m$ is found, we stop the search procedure; otherwise we pick up a flow demand uniformly at random (function `RandSelect`) from the flow demand list. At this time, the residual capacity of the subnetwork formed by current active nodes is not sufficient for carrying any new flow, thus more nodes are needed to be activated so that routing demands for the newly selected flow can be satisfied. Once a candidate flow d_c has been determined, we remove the incapable nodes (those that satisfy $\vec{S}_v \lessdot \vec{R}_c$ which means that these exists at least one dimension k such that $\vec{S}_v(k) > \vec{R}_c(k)$) according to the resources demand of the candidate flow and we denote by $\mathcal{G}_c$ the resulted network. Then, we apply a weight assignment process where we assign weights to the active nodes in $\mathcal{V}_a$ by invoking procedure `InvCount` (see below), and the weights for other nodes to be $(K(K-1)/2+1)$. In order to facilitate path selection, we carry out a node-link transformation procedure to assign weights for links based on the weights for nodes. The design of node weight assignment and node-link transformation will be detailed later in this section. At last, the candidate flow will be routed by involving a shortest-path-based algorithm such as Dijkstra algorithm on the weighted network $\mathcal{G}_c$ and will be removed from the demands list. The above process is repeated until the route for every flow has been assigned.

Inversion-based node weight assignment. We now describe the function `InvCount` for assigning weights to network

Algorithm 1 Multi-Resource Green (MRG) routing

$\mathcal{V}_a = \emptyset$; /*set of active nodes*/
for each $(v \in \mathcal{V})$ $\vec{S}_v = \{0\}^K$; /*residual resources*/
$\mathcal{E}_a \triangleq \{(v_1, v_2) \in \mathcal{E} \mid \forall v_1, v_2 \in \mathcal{V}_a\}$; $\mathcal{G}_a \triangleq \{\mathcal{V}_a, \mathcal{E}_a\}$;
while ($\mathcal{D}$ is not empty)
  $\mathtt{d}_c$ == none;
  for each $(\mathtt{d}_m \in \mathcal{D})$ /*Search for a candidate flow*/
    $\mathcal{G}_c^m = \mathcal{G}_a \setminus \{v \mid \vec{S}_v \lessdot \vec{R}_m\}$; /*remove incapable nodes*/
    if (IsConn($\mathcal{G}_c^m, v_m^{\mathrm{s}}, v_m^{\mathrm{t}}$) == true)
      $\mathtt{d}_c = \mathtt{d}_m$; $\mathcal{G}_c = \mathcal{G}_c^m$;
      break;
  if ($\mathtt{d}_c$ == none) /*candidate flow not found*/
    $\mathtt{d}_c = \mathtt{RandSelect}(\mathcal{D})$;
  $\mathcal{G}_c = \mathcal{G} \setminus \{v \mid \vec{S}_v \lessdot \vec{R}_c\}$;
  for each $(v \in \mathcal{V})$ /*node weight assignment*/
    if $(v \in \mathcal{V}_a)$ $w_v = \mathtt{InvCount}(\vec{S}_v, \vec{R}_m)$;
    else $w_v = K(K-1)/2 + 1$;
  for each $((v_1, v_2) \in \mathcal{E}_c)$ $w_e = (w_{v_1} + w_{v_2})/2$;
  $\mathcal{P}_c = \mathtt{SPath}(\mathcal{G}_c, \mathtt{d}_c)$; /*shortest path routing*/
  $\mathcal{V}_a = \mathcal{V}_a \cup \mathcal{P}_c$; $\mathcal{D} = \mathcal{D} \setminus \mathtt{d}_c$;
  for each $(v \in \mathcal{P}_c)$ $\vec{S}_v = \vec{S}_v - \vec{R}_m$;

nodes. The second observation we mentioned at the beginning of this section suggests that once we have obtained a candidate flow to route on the subnetwork comprised of the active capable nodes, it is important to decide which nodes are preferable to carry the candidate flow. We provide a measure based on the distributions of the load vectors of both the node residual capacities and the flow demand. The general notion is that if the resource dimensions of a node are all kept balanced, then more flows will likely fit into the node. As a consequence, the number of nodes that need to be active will be reduced. To clarify, we first introduce the concept of *inversion*.

Definition 2. *Given two vectors* $\vec{X} = \langle x_1, ..., x_n \rangle$ *and* $\vec{Y} = \langle y_1, ..., y_n \rangle$*, an* ***inversion*** *is defined as the condition* $x_i > x_j$ *and* $y_i < y_j$*,* $1 \leq i, j \leq n$*.*

Property 1. *Given two vectors in* n *dimensions, the total number of inversions is upper bounded by* $n(n-1)/2$*.*

As we are focusing on the distributions of the node residual capacities and the flow resources demands, it is straightforward that an inversion can lead to much heavier resource dimensions imbalance on a node as the scarce resource is demanded more and the abundant resource is demanded less. Therefore, in order to keep all the dimensions of resources balanced, the number of inversions has to be minimized. Based on this principle, the inversion-based node weight assignment procedure assigns weights for nodes that are already active according to the number of inversions shared by the node residual capacity vector and the flow demand vector. The weights of the inactive nodes are set to be one unit larger than the maximum number of inversions that can be shared by any residual capacity vector and flow demand vector. As a result, if possible, the nodes that are active and with less numbers of inversions will be preferably chosen to carry the candidate flow and the inactive nodes have the lowest priority to be used.

Algorithm 2 Hierarchical Green Routing (HGR)

function VBP($\mathcal{D}$) /*vector bin packing algorithm*/
  idx $= 1$; $\mathtt{d}_c =$ none; $S_{\mathrm{idx}} = \{0\}^K$;
  $\alpha_k = \sum_{\mathtt{d}_m \in \mathcal{D}} R_m(k) / \sum_{\mathtt{d}_m \in \mathcal{D}} \sum_{k=1}^{K} R_m(k)$;
  while ($\mathcal{D}$ is not empty)
    $\mathcal{D}_c = \mathcal{D} \setminus \{\mathtt{d}_m \in \mathcal{D} \mid S_{\mathrm{idx}} \lessdot R_m\}$;
    $\mathtt{d}_c = \arg\min_{\mathtt{d}_m \in \mathcal{D}_c} \sum_{k=1}^{K} \alpha_k (S_{\mathrm{idx}}(k) - R_m(k))^2$;
    if ($\mathtt{d}_c$ == none) idx++; /*open a new bin*/
    else $\mathcal{D} = \mathcal{D} \setminus \{\mathtt{d}_c\}$; /*pack the current item*/
  return idx
for $(0 \leq i \leq z-1)$ /*# of aggr. nodes in each pod*/
  $N_i^{\mathrm{agg}} = \mathtt{VBP}(\{\mathrm{d}_m \mid v_m^{\mathrm{s}}$ or v_m^{t} in pod $i\})$
for $(0 \leq j \leq z/2-1)$ /*# of core nodes*/
  $N_j^{\mathrm{core}} = \mathtt{VBP}(\{\mathrm{d}_m \mid (v_m^{\mathrm{s}}$ or $v_m^{\mathrm{t}} \bmod (z^2/4))/2 = j\})$

Path selection. The path selection process is equivalent to solving a node-weighted single-source shortest path routing problem. We notice that this problem can be transformed into a traditional link-weighted single-source shortest path routing problem by setting the weight of each link to be the half of the sum of the weights of the endpoints of this link. Denote by $\mathbb{R}_1$ and $\mathbb{R}_2$ the node-weighted and the transformed link-weighted shortest path routing problems respectively. We have

Property 2. *Solving* $\mathbb{R}_1$ *is equivalent to soving* $\mathbb{R}_2$*.*

As a result, solving the corresponding link-weighted single-source shortest path routing problem will also give solutions to the path selection for the candidate flow. It is well-known that the link-weighted single-source shortest path routing problem can be solved efficiently by using the Dijkstra algorithm.

C. Time Complexity

The algorithm runs iteratively and the total number of iterations will be upper bounded by the number of flow deamands M. In each iteration, the dominant time consumer is the depth-first search for the candidate flow searching procedure, which can be accomplished in $\mathcal{O}(|E|)$ time where $|E| \leq N^2$ is the total number of edges in the network (N is the total number of nodes). Therefore, the MRG algorithm takes $\mathcal{O}(|E|M^2)$ time.

IV. Topology-aware Heuristic

The proposed MRG algorithm can leverage the coordination of the flow demands in multiple dimensions and minimize the number of active network nodes efficiently. However, MRG is generally conducted without taking into account the topology features of the network. We notice that topologies of the networks commonly used in data center networks such as fat-tree or VL2 have very high level of symmetry and they are usually well structured in layers. Therefore, we argue that the routing algorithm can be further improved by taking advantage of the topology characteristics. In this section, we provide a

new topology-aware heuristic for the most common tree-like data center network topologies.

The key observation we have from tree-like topologies is that the number of active nodes can be determined layer by layer. We take a typical fat-tree topology as an example. The number of edge nodes cannot be optimized since edge nodes are also responsible for inter-host communication in the same rack. In each pod, the number of aggregation nodes can be determined according to the flow demands that flow out of and into the pod. This is actually to solve a vector bin packing problem as we have introduced previously. The core layer is a bit different from the aggregation layer; for a z-ary fat-tree, all the cores nodes that share congruence with respect to $(z/2)$ will be responsible for carrying the flow demands from the aggregation nodes in the same positions in every pod. Thus for these core nodes, solving a vector bin packing can give the right number of nodes that need to stay active. Inspired by this observation, we propose HGR, a hierarchical energy-efficient routing algorithm based on solving a set of vector bin packing problems. The pseudocode of HGR is shown in Algorithm 2.

Vector bin packing. The function VBP we adopted for solving the vector bin packing problem is a norm-based greedy algorithm [16]. The algorithm is bin-centric which means that it focuses on one bin idx and always places the most suitable remaining item that fits in the bin. To find out the most suitable item, the algorithm looks at the difference between the demand vector R_m and the residual capacity vector S_{idx} under a certain norm. We choose the ℓ_2-norm and from all unassigned items, we choose the item that minimizes $\sum_{k=1}^{K} \alpha_k (S_{\mathrm{idx}}(k) - R_m(k))^2$ where α_k represents the importance of dimension k among all dimensions and is given by

$$\alpha_k = \frac{\sum_{\mathsf{d}_m \in \mathcal{D}} R_m(k)}{\sum_{\mathsf{d}_m \in \mathcal{D}} \sum_{k=1}^{K} R_m(k)}.$$

If no item can be found to fit into the current bin idx, we open a new bin and repeat the above procedure.

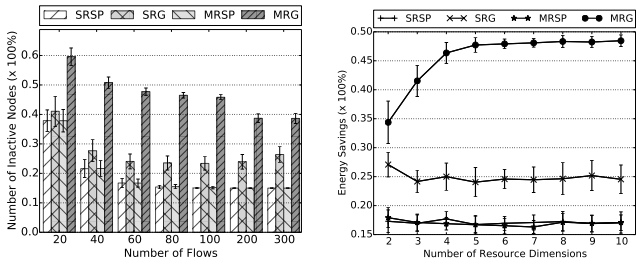
Time complexity. The HGR algorithm replies on solving several instances of the vector bin packing problem. In the worst case, the sizes of the vector bin packing instances can be as large as $\mathcal{O}(M)$ and thus it will take $\mathcal{O}(M^2)$ time to be solved by VBP algorithm. As a result, the total time complexity of HGR can be given by $\mathcal{O}(M^2)$. Compared to the MRG algorithm, HGR can provide a speedup of $\Omega(|E|)$. We will validate this speedup by simulations.

V. Evaluation

A. Simulation Settings

We deploy our algorithms on a laptop with a Core i5 2.6GHz CPU with two physical cores and 8GB DRAM. All of the algorithms are implemented in Python.

We choose fat-trees of different sizes as the data center network topologies. This is because fat-tree is a typical topology used in DCNs, and can provide equal-length parallel paths between any pair of end hosts, which is very beneficial for software packet processing paradigm to embed processing functions into the routing paths regardless of the topology details. The flow demands we used in our simulations are generated randomly: the endpoints of each flow are chosen uniformly at random from the set of end hosts. The requirement of each resource dimension of each flow is generated following a normal distribution (in the positive side) where the mean and the variation are all set to be 0.02 to provide large resource demand diversity. The node capacity of each resource dimension is assumed to be normalized to 1.



(a) Energy savings under different numbers of flows

(b) Energy savings under different numbers of resource dimensions

Figure 1. Performance comparison for MRG under the scenarios where the network topology is given by an 8-ary fat-tree with 208 nodes (128 end-hosts and 80 packet processors).

We carry out two groups of simulations for validating MRG and HGR respectively: i) For evaluating the performance of algorithm MRG, we compare it with three other algorithms of interest: Single-Resource Shortest Path (SRSP), Single-Resource Green (SRG), Multi-Resource Shortest Path (MRSP). The efficiency of energy saving of the four algorithms are examined on two fat-tree topologies in different scales under different numbers of flow demands. We also explore the impact of the number of resource dimensions under certain scenarios. ii) The performance of algorithm HGR is compared with that of algorithm MRG. We first study the impact of the number of resource dimensions. Then, under certain scenarios, we examine the efficiency of energy saving and the running time of both MRG and HGR under different numbers of flow demands. All the results are averaged among 20 independent tests and all the figures show with the average and the standard deviation.

B. Performance of Algorithm MRG

Energy savings. The simulation results for evaluating the energy saving performance of MRG are depicted in Fig. 1(a, b, c). The energy saving ratio is represented by the number of inactive nodes divided by the total number of nodes. It can be seen from Fig. 1(a) that MRG outperforms the other three algorithms with respect to energy savings under all scenarios. SRSP and MRSP converge to very low energy saving ratios very quickly while SRG and MRG can exploit more energy saving potentials by carefully steering traffic flows. We also compare the performance of all the algorithms under extremely heavy load scenarios. When the number of flow demands exceeds the capability of the network (and congestion happens at some critical nodes), MRSP and MRG will block more flows than SRSP and SRG. This is reasonable because MRSP and

Table I
RUNNING TIME STATISTICS OF THE ALGORITHMS (UNIT: SEC)

# of flows	20	40	60	80	100	120
alg. MRG	5.37	16.63	37.00	58.26	92.63	101.89
alg. HGR	0.026	0.078	0.192	0.400	0.647	0.681

MRG take into account more resource dimensions and it is likely that node capacities are violated more easily than with single-resource solutions. However, when considering only one resource dimension, some nodes will be congested due to the neglect of other resource dimensions, although more flow demands are likely to be assigned.

Impact of the number of resource dimensions. Fig. 1(b) depicts the simulation results for examining scalability of MRG with respect to the number of resource dimensions. It can be obviously noticed that the energy saving performance of MRG has a very significant improvement with the increase of the number of resource dimensions and converges to a high level. This is because with more resource dimensions, the proposed inversion-based node weight assignment can distinguish nodes from one another more accurately and thus the path chosen for each flow will be more effective in terms of energy saving.

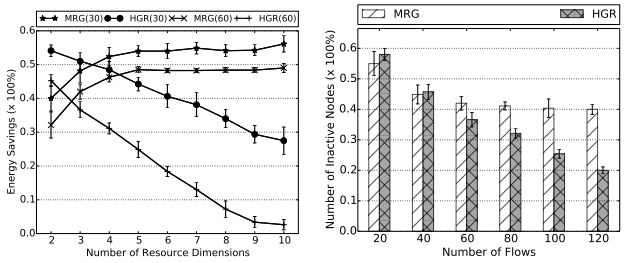
C. Performance of Algorithm HGR

We first compare the scalability of MRG and HGR with respect to the number of resource dimensions. The simulation results are shown in Fig. 2(a). It can be observed that HGR outperforms MRG when the number of resource dimensions is very small. However, with the increase of the number of resource dimensions, the energy saving performance of HGR drops dramatically with a constant rate, while MRG performs better and better and converges finally as we have discussed before. This is mainly because HGR is largely based on the vector bin packing heuristic which performs well when the number of dimensions is small due to the greedy manner of item assignment, but it has limited scalability with respect to the number of dimensions.

We then choose a fair number of resource dimensions ($K = 3$) and compare both the energy saving ratio and the running time of MRG and HGR. The energy saving results are depicted in Fig. 2(b). We observe that when the number of flow demands is not very large, MRG and HGR are comparable in terms of energy savings, but HGR suffers from some performance degradation when the number of flows is very large. However, HGR compensates this slight loss of energy efficiency by a very significant reduction on the running time. As can be seen from Table I, for a fat-tree with 80 packet processing nodes (i.e., $|E| = 192$), the running time of HGR is around 0.5 percent of that of MRG, which confirms the lower bound on the speedup $\Omega(|E|)$.

VI. CONCLUSION

We study the energy-efficiency multi-resource routing problem which arises from the recently proposed cloud networking model NaaS. This optimization problem differs from the traditional energy-efficient routing problem by having node capacities and flow demands represented by vectors in multiple dimensions. We provide a simple iterative routing scheme which selects flows iteratively to exhaust the residual capacities in active nodes and assign routes to flows based on the distributions of node residual capacities and flow demands. To leverage the structural property of data center network topologies, we also provide a topology-aware heuristic designated to fat-trees, which can provide comparably good energy efficiency while significantly reducing the computation time.



(a) Energy savings under different numbers of resource dimension (b) Energy savings under different numbers of flows

Figure 2. Performance comparison for HGR under the scenarios where the network topology is given by an 8-ary fat-tree with 208 nodes (128 end-hosts and 80 packet processors).

REFERENCES

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *NSDI*, 2010, pp. 281–296.

[2] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: fine grained traffic engineering for data centers," in *CoNEXT*, 2011, p. 8.

[3] OpenFlow. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[4] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "Cloudnaas: a cloud networking platform for enterprise applications," in *SoCC*, 2011, p. 8.

[5] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "Naas: Network-as-a-service in the cloud," in *Hot-ICE*, 2012.

[6] P. Costa, A. Donnelly, A. I. T. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications," in *NSDI*, 2012, pp. 29–42.

[7] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *NSDI*, 2010, pp. 249–264.

[8] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in *INFOCOM*, 2012.

[9] L. Wang, F. Zhang, J. A. Aroca, A. V. Vasilakos, K. Zheng, C. Hou, D. Li, and Z. Liu, "Greendcn: A general framework for achieving energy efficiency in data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 4–15, 2014.

[10] L. Wang, A. Fernández Anta, F. Zhang, J. Wu, and Z. Liu, "Multi-resource energy-efficient routing in cloud data centers with network-as-a-service," 2015, arXiv:1501.05086 (http://arxiv.org/abs/1501.05086).

[11] N. Bansal, A. Caprara, and M. Sviridenko, "Improved approximation algorithms for multidimensional bin packing problems," in *FOCS*, 2006.

[12] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *SIGCOMM*, 2012, pp. 1–12.

[13] M. Andrews, S. Antonakopoulos, and L. Zhang, "Minimum-cost network design with (dis)economies of scale," in *FOCS*, 2010.

[14] R. Krishnaswamy, V. Nagarajan, K. Pruhs, and C. Stein, "Cluster before you hallucinate: approximating node-capacitated network design and energy efficient routing," in *STOC*, 2014, pp. 734–743.

[15] G. J. Woeginger, "There is no asymptotic ptas for two-dimensional vector packing," *Inf. Process. Lett.*, vol. 64, no. 6, pp. 293–297, 1997.

[16] R. Panigrahy, K. Talwar, L. Uyeda, and I. Wieder, "Heuristics for vector bin packing," in *ESA*, 2011.