Computer Networks 139 (2018) 70–80

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A spectral clustering approach to network-aware virtual request partitioning[☆]

Lingnan Gao[*], George N. Rouskas

North Carolina State University, Raleigh, NC 27695-8206, USA

ARTICLE INFO

Article history:
Received 20 December 2016
Revised 2 April 2018
Accepted 3 April 2018
Available online 5 April 2018

ABSTRACT

Virtual request partitioning is an essential subproblem of two common problems in virtual networks, namely, virtual network embedding (VNE) and virtual machine placement (VMP). In this study, we consider a network-aware variant of the problem where the objective is to partition a virtual request so as to minimize the total amount of inter-cluster traffic. This problem is equivalent to the (k, v)-balanced partitioning problem, an NP-complete problem. To handle the inherent complexity of this problem, we develop a spectral clustering-based partitioning scheme that produces good solutions in a reasonable amount of time. Our solution consists of several components: (a) spectral clustering, (b) a Constrained k-means partitioning algorithm that ensures that capacity limits for clusters are met, and for which we present a polynomial-time greedy algorithm, and (c) a greedy refinement algorithm using simulated annealing to further improve the clustering solution. Simulation results indicate that our algorithm outperforms existing partitioning schemes in terms of inter-cluster traffic minimization.

1. Introduction

Network virtualization is seen as crucial way in reshaping the Internet architecture and introducing diversity into the current network [1]. With network virtualization, conventional providers are decoupled into infrastructure providers (InP), who mainly focus on the management of the infrastructure, and service providers (SP), who are responsible for the creation of the network and provide end-to-end service to end users. Such an environment allows the deployment of network architectures regardless of the underlying infrastructure, and thus facilitates the evolution of network architecture [2]. The cloud computing paradigm also employs virtualization techniques. Data centers aggregate all the computing resources (including CPU, memory, and storage), and provide services to the end users in the form of virtual machines (VM). Server virtualization allows multiple VMs to co-locate on the same physical server to increase utilization and lower the operational cost [3].

A key challenge for network virtualization and cloud computing is resource allocation. In network virtualization, resource allocation arises in the context of the virtual network embedding (VNE) problem, where the objective is to embed the virtual network to the substrate network so as to maximize the benefit from the existing hardware [4]. In the area of cloud computing architecture, the related virtual machine placement (VMP) problem arises, whereby the objective is to optimally assign the VMs to physical hosts so as to utilize the available resources without performance degradation [5].

In either area, mapping virtual request to physical resources may involve partitioning of the virtual request. For the VNE problem, mapping virtual requests to multiple domains may be required for various reasons, including load balancing [6] and managing the embedding cost [7]; for the VMP problem, VMs must be mapped onto underlying physical resources that may span across physical hosts, racks, even data centers [8]. Therefore, for communication-intensive applications, mapping virtual requests to physical resources must be accomplished in a manner that satisfies capacity constraints and takes into account the communication cost and quality of service (QoS) requirements [8,9].

In this work, we consider the problem of virtual request partitioning and present an algorithm inspired by spectral clustering to partition the set of virtual nodes under capacity constraints. This algorithm produces high quality solutions, compares favorably to existing algorithms, and scales well; simulation experiments indicate that it can tackle virtual networks consisting of hundreds of nodes within a few seconds. Following the introduction, we review previous work in this topic in Section 2. In Section 3, we formally define the problem and present the various components of the virtual request partitioning algorithm. In Section 4, we present the results of simulation experiments we have conducted to compare

[☆] This work was supported in part by the National Science Foundation under Grant CNS-1111088.

[*] Corresponding author.
E-mail address: glingna@ncsu.edu (L. Gao).

https://doi.org/10.1016/j.comnet.2018.04.005
1389-1286/© 2018 Elsevier B.V. All rights reserved.

the performance of this algorithm to existing algorithms. We conclude the paper in Section 5.

2. Related work

Several studies [6,7,9] have addressed the virtual request partitioning problem using max-flow, min-cut schemes. With existing algorithms, it is possible to compute efficiently the maximum flow between a pair of nodes and obtain the minimum cut between them. The work in [7] recursively uses the max-flow, min-cut approach to partition the network into the desired number of clusters. In [6,9], a clustering approach based on Gomory–Hu trees is explored. A Gomory–Hu tree represents the $n-1$ minimum $s-t$ cuts in a graph of n nodes. By removing the $k-1$ least weight edges of this tree, a partition of the n nodes into k clusters is obtained that is close to optimal. The shortcoming of this approach is that the resulting clusters may be highly imbalanced, as the cluster capacity is not taken into account. In order to enforce the capacity constraints, further partitioning of an overloaded cluster and combination of small clusters is necessary. For instance, in an extreme case, when recursively using the max-flow min-cut approach to partition the network requests, one may keep obtaining the result of one node in one part while all the rest goes to the other; while for a Gomory–Hu tree, it is possible to end up with a star topology, and by removing one edge, each time, we can only obtain a cluster with a single node. However, there is no guarantee that the combination of those small clusters would lead to a small amount of inter-cluster traffic. In both cases, when we group those singleton nodes into a cluster, there is no evidence that the traffic among those nodes is high. As intra-cluster traffic is not necessarily low, this, in turn, implies a potentially high inter-cluster traffic.

The virtual network embedding problem across multiple domains has been considered in [10], where it was proposed to use iterative local search (ILS) to partition the virtual request. For this problem, ILS starts with a random clustering, following which a sequence of solutions is generated by randomly remapping some of the nodes to other clusters. Of these solutions, the one that improves upon the current solution the most is kept, and the algorithm iterates until a stopping criteria is met. Despite the simplicity of this method, it is hard to guarantee the quality of the solution within a limited time. In a related study, a general procedure for resource allocation in distributed clouds was presented in [8]. The objective was to select the data centers, the racks, and processors with the minimum delay and communication costs, and then to partition the virtual nodes by mapping them onto the selected data center and processors.

In [25], a series of spectral partitioning algorithms are reviewed and summarized. These spectral partitioning techniques generally use the median of the Fiedler vector, the second smallest eigenvalue of the Laplacian matrix for the traffic matrix, to partition the graph into two parts. Then, by recursively applying this method, one can partition the graph into $2^n, n \geq 1$, parts. These algorithms have two limitations. First, the node weights related to load demands are not taken into consideration; consequently, the load may not be balanced well across the various clusters. Second, the number of clusters is limited to powers of two. In [26], a spectral partitioning based method that makes use of multiple eigenvectors of the Laplacian matrix is proposed. While this work considers the node weight balancing and makes use of multiple eigenvectors to produce a solution with a high-quality, it only partitions a graph into two, four or eight parts.

In contrast to the existing works, our algorithm exploits multiple eigenvectors of the Laplacian matrix to partition the network requests into arbitrary number of clusters. Such eigenvectors would form a k-dimensional space known as the eigenspace, and the Euclidean distance would reflect the traffic intensities. One can arrive at a high quality solution by clustering on Euclidean distance. Unlike Gomory–Hu tree based approaches, while assigning data points to the different partitions, the minimization of the inter-cluster traffic and the cluster capacities are jointly considered. This allows network requests to be partitioned into arbitrary clusters under the capacity constraints.

3. Virtual request partitioning

Virtual request partitioning is required in both the VNE and VMP problems, whereby the objective is to partition the virtual network into a set of clusters in order to minimize the inter-cluster traffic. Fig. 1(a) shows a set of virtual nodes that have been partitioned into three clusters such that inter-cluster traffic is minimum. In the VNE scenario of Fig. 1(b), mapping each of the clusters to a different domain will minimize inter-domain traffic (which presumably is more expensive than intra-domain traffic). In the context of the VMP problem in Fig. 1(c), assuming that each cluster is assigned to a different processor or even rack, optimal partitioning of the virtual request minimizes the traffic that has to be handled by the aggregate and core switches of the data center network, hence improve the scalability and stability of the network.

In this section, we formally define the virtual request partitioning problem which could be applied both to the VNE and VMP problem, and then present an algorithm based on spectral clustering to solve this problem.

3.1. Problem statement

We model the communication between virtual nodes as a traffic matrix $W = [w_{ij}]_{n\times n}$, where element w_{ij} represents the amount of traffic from virtual node i to j. Each virtual node is associated with a resource requirement r_i, and each cluster h is associated with a capacity threshold Cap_h.

With these definitions, partitioning the set of virtual nodes into k clusters so as to minimize the inter-cluster traffic can be formulated as the following Integer Linear Programming (ILP) problem:

$$\textbf{minimize} \quad \sum_k \sum_{i,j} w_{ij}(1 - y_{ij}^k) \tag{1}$$

$$\textbf{subject to} \quad \sum_i r_i x_i^k \leq Cap_k, \qquad \forall k \tag{2}$$

$$x_i^k + x_j^k \leq y_{ij}^k + 1 \qquad \forall i, j, k \tag{3}$$

$$y_{ij}^k \leq x_i^k \qquad \forall i, j, k \tag{4}$$

$$\sum_k x_i^k = 1, \qquad \forall i \tag{5}$$

$$x_i^k = \{0, 1\}, y_{ij}^k = \{0, 1\} \tag{6}$$

The binary variable $x_i^k \in \{0, 1\}$ here indicates if virtual node i is assigned to cluster k while binary variable $y_{ij}^k \in \{0, 1\}$ indicates if virtual nodes i and j are both mapped onto cluster k.

Constraint (2) ensures that the amount of resources assigned to each cluster will not exceed its capacity limit. Constraint (3) and constraint (4) guarantees consistency between decision variable x and y. Constraint (5) makes sure that virtual machine i is assigned to exactly one cluster. This formulation is equivalent to the (k, v)-balanced partitioning problem, which is an NP-complete problem [16]. We also note that by replacing “virtual node” with “VM” and cluster with “processor,” the above formulation also expresses the

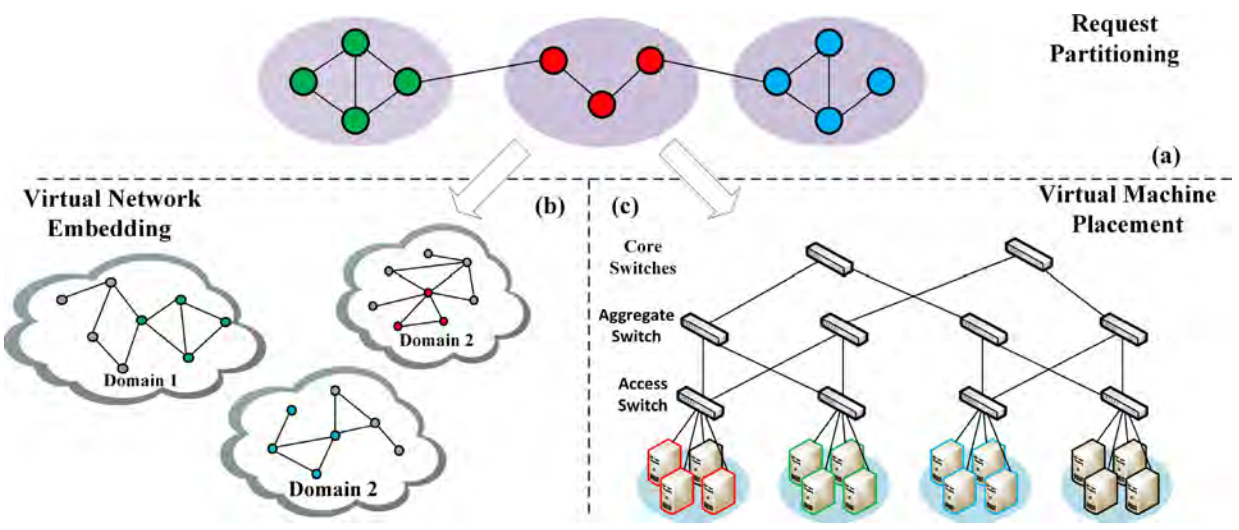

Fig. 1. Virtual request partitioning for virtual network embedding and virtual machine placement.

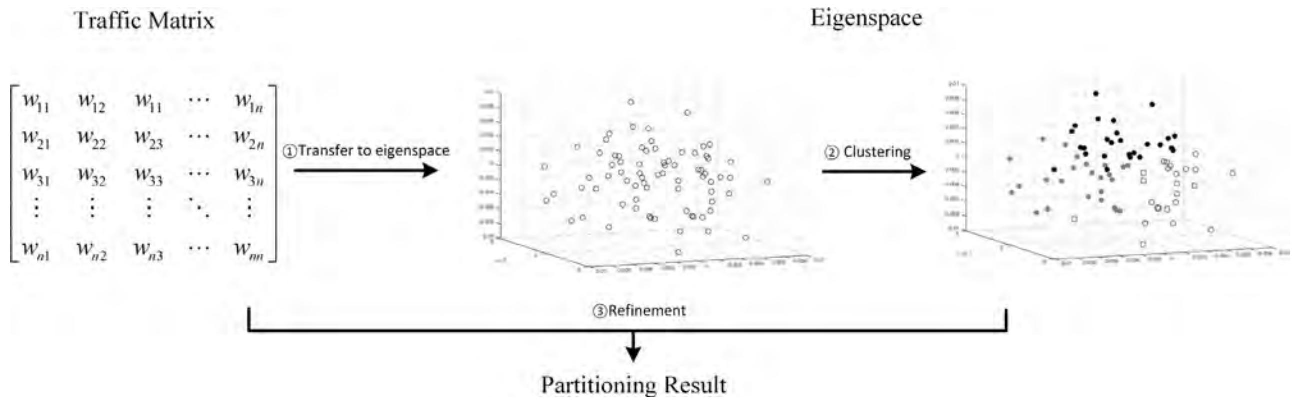

Fig. 2. General procedure for virtual request partitioning.

problem of placing VMs onto processors so as to minimize inter-processor traffic.

For the solving of the virtual request partitioning problem, we apply a spectral clustering based [11] approach. A high level idea of our approach is illustrated in the Fig. 2. We aim to partition the virtual request that consists of n nodes. Starting with a n-by-n traffic matrix W, we compute the normalized Laplacian matrix L_{sym} of the given traffic matrix W. We then compute the eigenvectors that associate with the first k smallest eigenvalues of matrix L_{sym}, and form a n-by-k matrix Z based on the k-smallest eigenvectors. Denote ith row of the matrix, a k dimensional vector, as z_i. The virtual node i will be associated with z_i. We can think of the z_i as the coordinates for the virtual node i on a k dimensional space, and the matrix Z as the collection of coordinates for the n data points. This k-dimensional space is referred as Eigenspace. With given property of the Laplacian matrix, the nodes that have higher traffic among them tend to stay closer on this Eigenspace. Thus, we can perform clustering for those data points $(z_i)_{i=1,\dots,n}$. Clustering will assign the nodes to different clusters so as to minimize the total distance between the nodes within each cluster on the Eigenspace, which implies this assignment maximizes the intra-cluster traffic and eventually minimize the inter-cluster traffic. In order to satisfy the capacity constraints, we perform Constrained k-means algorithm for clustering. Upon the completion of the clustering, we obtain our initial clustering results. The result can be further refined using the Greedy-Refinement algorithm based on the traffic matrix W. It will iteratively assign the Virtual Node to another cluster if such an assignment would lower the inter-cluster traffic. And to achieve a better partitioning result, we can combine the Greedy Refinement algorithm with Simulated Annealing. The whole procedure is outlined as Algorithm 1, and is explained in detail in subsequent sections.

Algorithm 1 Spectral clustering based Virtual Request Partitioning Algorithm

```
Input:
    $W$: $n \times n$ traffic matrix of VNodes
    $k$: number of clusters
    $R = r_1, r_2, \dots, r_n$: resource requirement of VNodes
    $Cap = cap_1, cap_2, \dots, cap_k$: capacity of each cluster
Output:
    The cluster to which each VNode belongs
1: Construct diagonal matrix $D$ with $d_{ii} = \sum_{j=1}^{n} w_{ij}$
2: Compute normalized Laplacian $L_{sym} = D^{-1/2}(D - W)D^{-1/2}$
3: Obtain the eigenvector associated with the $k$ smallest eigenvalues of matrix $L_{sym}$
4: Let matrix $U$ contain the above eigenvectors as columns
5: Let $z_i$ be the vector associated with $i^{th}$ row of $U$.
6: Cluster the points $(z_i)|_{i=1\dots n}$ under the capacity constraints $Cap$ via **Constrained $k$-means**
7: Refine the partitioning result by **Greedy-Refinement** based Simulated Annealing
```

3.2. Spectral clustering

Spectral clustering [11] is used to find a set of clusters such that the edges between clusters have low weights (in this case, the weights would represent the pairwise traffic). An important feature of spectral clustering is that, unlike the conventional min-max flow approach, it can avoid the creation of imbalanced partitions whereby some clusters are assigned a much larger number of nodes than others. Given a $n \times n$ traffic matrix W, the normalized Laplacian matrix is defined as $L_{sym} = D^{-1/2}(D - W)D^{-1/2}$, where D is a diagonal matrix with element $d_{ii} = \sum_j^n w_{ij}$.

Let $P_1, \ldots, P_k$ be a partition of the set of n virtual nodes into k sets (clusters), i.e., the sets P_i are pairwise disjoint and their union is $\{1, \ldots, n\}$. Further, let $\bar{P}_i$ be the complement of set P_i. We define the $N\,Cut$ metric as:

$$N\,Cut(P_1, P_2, ..., P_k) = \sum_{i=1}^{k} \frac{cut(P_i, \bar{P}_i)}{vol(P_i)} \tag{7}$$

where the numerator represents inter-cluster traffic (i.e., between nodes in P_i and nodes not in P_i) and the denominator denotes total traffic within cluster P_i.

Minimizing $N\,Cut$ will result in a set of k clusters that have low inter-cluster traffic, while the presence of $vol(P_i)$ in the denominator will prevent the creation of clusters with few nodes, and hence, cluster sizes will not be highly imbalanced. The normalized Laplacian has the following property that allows us to find an approximate solution to the $N\,Cut$ problem efficiently: for a given n-by-k matrix H, if we take h_{ij} as:

$$h_{ij} = \begin{cases} 1/\sqrt{vol(P_i)} & \text{if } v_i \in P_j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

the following equation would hold [11]: $Tr(H^T L H) = \sum_{i=1}^{n} \frac{cut(P_i, \bar{P}_i)}{|vol(P_i)|} = N\,Cut(P_1, P_2, ..., P_k)$.

Also observe that $H^T D H = I$. Therefore, we can reformulate the problem of minimizing $N\,Cut$ as follows:

$$\begin{aligned} \textbf{minimize} \quad & Tr(H^T L H) \\ \textbf{subject to} \quad & H^T D H = I \\ & h_{ij} = \{1/\sqrt{vol(P_i)}, 0\} \end{aligned} \tag{9}$$

We can obtain an approximate solution to this problem in polynomial time by relaxing the last condition and taking $Z = D^{-1/2}H$. The problem then becomes:

$$\begin{aligned} \textbf{minimize} \quad & Tr(Z^T L_{sym} Z) \\ \textbf{subject to} \quad & Z^T Z = I \end{aligned} \tag{10}$$

According to the Rayleigh–Ritz Theorem, the solution to problem (10) would be to take Z as the k smallest eigenvectors of L_{sym}, i.e., the eigenvectors corresponding to the k smallest eigenvalues.

Let matrix Z be a n-by-k matrix that contains the above k eigenvectors as columns, and let z_i be the vector associated with the ith row of Z, and we take z_i as coordinates for the virtual nodes on the Eigenspace. The data points will stay close to each other in the Eigenspace if the traffic between them is high, and this will allow us to obtain the final solution using clustering algorithm. A formal proof of this property that established in the matrix perturbation theory is presented in [19].

To obtain the final solution, clustering will be performed to cluster the points $(z_i)_{i=1,\ldots,n}$ while satisfying the capacity constraints Cap. Our approach to clustering is the topic of the following subsections.

3.3. Constrained k-means

Conventional spectral clustering uses the k-means algorithm [18] to cluster the data points z_i. One drawback of the k-means algorithm is that it may converge to a solution in which some clusters have very few data points while others are overloaded. Therefore, we use the Constrained k-means algorithm proposed in [12]. Given a set of data points, the Constrained k-means algorithm aims to find a set of cluster centers $C_1, C_2, \ldots, C_k$, such that the sum of distances between each node and the center it is assigned to is minimal. Specifically, the problem solved in [12] is:

$$\begin{aligned} \textbf{minimize} \quad & \sum_{i=1}^{m} \sum_{h=1}^{k} x_i^h \cdot \left(\frac{1}{2}\|z_i - C_h\|_2^2\right) \\ \textbf{subject to} \quad & \sum_{h=1}^{k} x_i^h = 1, \ \forall i; \quad x_i^h \geq 0, \ \forall i, \ \forall h. \\ & \sum_{i} x_i^h \geq \mathcal{T}_h \forall h \end{aligned} \tag{11}$$

In this formulation, x_i^h is a selection variable denoting whether data point i belongs to cluster h. The last constraint is used to control the size of each cluster, i.e., to ensure that each cluster has size at least equal to $\mathcal{T}_h$.

In the virtual request partitioning problem, the resource requirement for each cluster should not exceed its capacity. To this end, we replace the constraint $\sum_i x_i^h \geq \mathcal{T}_h$ with $\sum_i r_i x_i^h \leq Cap_h$, and follow the iterative method proposed in [12].

Given n data points z_1, z_2, ..., z_n, k cluster center points $C_1^t, C_2^t, ..., C_k^t$ at iteration t, and capacity limit Cap_h for cluster h, the cluster center for iteration $t+1$ is computed using the following steps.

Cluster Assignment. Given the fixed cluster center points C_h, find the selection variables so that the distance between the data points and the corresponding cluster center is minimized.

$$\begin{aligned} \textbf{minimize} \quad & \sum_{i=1}^{n} \sum_{h=1}^{k} x_i^h \cdot \left(\frac{1}{2}\|z_i - C_h\|_2^2\right) \\ \textbf{subject to} \quad & \sum_{h=1}^{k} x_i^h = 1, && \forall i \\ & \sum_{i} r_i x_i^h \leq Cap_h, && \forall h \\ & x_i^h \geq 0, && \forall i, \forall h \end{aligned} \tag{12}$$

Cluster Update. Compute the center point at iteration $t+1$ as:

$$C_h^{t+1} = \begin{cases} \frac{\sum_{i=1}^{n} x_i^h(t) z_i}{\sum_{i=1}^{n} x_i^h(t)} & \text{if } \sum_{i=1}^{n} x_i^h(t) > 0 \\ C_i^t & \text{otherwise} \end{cases} \tag{13}$$

It was shown in [13] that cluster assignment is equivalent to the Minimal Cost Flow (MCF) problem. We now show that this cluster assignment subproblem can be solved optimally within $O(kn \log n)$ time using a greedy approach.

We first reduce cluster assignment to the MCF problem following the steps outlined in [13]. The supply from the source node (src) and the demand by sink node (dst) is equivalent to the total requirement $\sum_{i=1}^{n} r_i$. src is connected to all the data points $(z_i)|_{i=1,\ldots,n}$, and each data point is connected with all the cluster centers, while cluster centers are connected to dst. Each edge is associated with a weight tuple (*Price, MaximumCapacity, Flow*). The *Price* from data points to cluster centers are set to the corresponding distance, while on other edges it is set to zero. The *Maximum-Capacity* from src to the data points is the resource requirement r_i and from cluster center h to dst is Cap_h; on other edges, it is set to

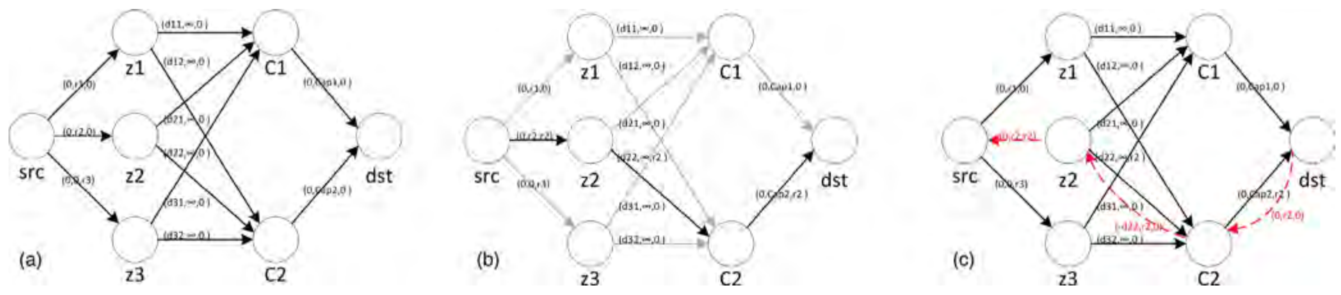

Fig. 3. MCF view of clustering assignment subproblem.

infinity. An example of the representation of a cluster assignment problem to an MCF network is shown in Fig. 3(a).

Now, we show how this problem can be solved with a greedy approach. First, ascending sort the price on all the paths from *src* to *dst* and augment flow accordingly. When we think of this problem in terms of negative cycle canceling algorithm, each time we augment the flow by f on a path, we create a reverse path with negative price on the residual graph. An example can be found in Fig. 3 (b) and (c).

A brief proof of optimality is as follows. Denote the iteration to augment flow on path i as t_i. At t_i, we augment flow on path i. For $t_j > t_i$, no negative cycle will form on the residual graph involving the reversed path i, because the price of path j will be no less than of path i. Also, for path j with $t_i > t_j$, the price of path i is higher, hence a negative cycle will be formed only when we take forward direction from on path j and backward on i, which is impossible. This is from the fact on path j, the capacity on *src* to a data point or from cluster center of *dst* is depleted. In the former case, we cannot find a forward path from *src* to the data point, so path j is blocked, and no cycle will form. The same applies to the latter case when path i and j go through a different cluster center. If they pass through same cluster center, then, we cannot augment the flow on path i, because there is no available capacity from the cluster center to *dst*, so no negative circle will form. In conclusion, no negative cycle can be found and the solution will be optimal.

At each step, denote the remaining capacity of cluster h as Rm^h_{cap}, and the remaining resource requirement of each virtual node i as Rm^i_{res}. Our Constrained k-means with greedy cluster assignment algorithm is shown as Algorithm 2.

Note that some of the resulting variables may be fractional. We round the fractional selection variable by assigning the node i to the cluster h with maximum x^h_i without violating the capacity constraints. Observe that the number of fractional elements will be smaller than the number of clusters, because each element will not be fragmented unless the cluster it is assigned to reaches its maximum capacity; after that instant, the cluster will take no additional data points. As a result, no other assignment will get fragmented on that cluster, and therefore, the number of fractional assignment will be less than that of clusters. Assuming that $n \gg k$, i.e., the number of data points is significantly greater than the number of clusters, we expect that this greedy rounding scheme will have only relatively small negative impact.

Algorithm 2 Constrained k-Means

```
Input:
    $(z_i)_{i=1,...,n}$: data points formed by eigenvectors
    $k$: number of clusters
    $R = r_1, r_2, ..., r_n$: resource requirement of VNodes
    $Cap = cap_1, cap_2, ..., cap_k$: capacity of each cluster
Output: Selection indicator $x^h_i$
 1: Iteration $t \leftarrow 0$, randomly initialize $C^t_h\ \forall h$
    Remaining resource requirement $Rm_{req} \leftarrow R$
    Remaining capacity $Rm_{cap} \leftarrow Cap$
 2: while $C^{t+1} \neq C^t$ do
 3:     Clustering Assignment:
 4:     Compute pairwise distance between data points and cluster
        centers $D = \{d_{11}, d_{12}, ..., d_{nk}\}$
 5:     Ascending sort $D$ to get $D_{asc} = \{d'_1, d'_2, ..., d'_{n*k}\}$
 6:     for $j \leftarrow 1$ to $(n*k)$ do
 7:         Choose point $i$ and center point $h$ associated with $d'_j$
 8:         if $Rm^i_{req} < Rm^h_{cap}$ then
 9:             $x^h_i \leftarrow Rm^i_{req}/r_i$; $Rm^h_{cap} \leftarrow Rm^h_{cap} - R^i_{res}$; $Rm^i_{res} \leftarrow 0$
10:         else
11:             $x^h_i \leftarrow Rm^h_{cap}/r_i$; $Rm^i_{req} \leftarrow Rm^i_{req} - Rm^h_{cap}$; $Rm^h_{cap} \leftarrow 0$
12:         end if
13:     end for
14:     Clustering Update:
15:     update the center points according to (13)
16:     $t \leftarrow t+1$
17: end while
18: $P \leftarrow$ round $S$ without violating capacity constraint.
```

3.4. Partitioning refinement

In order to improve upon the partition results obtained by the Constrained k-means algorithm, we employ a Simulated Annealing (SA) based approach [27]. The SA algorithm aims to approximate an optimal solution by probabilistically accepting a new lower quality solution. It mimics the annealing process, and starting with a high initial temperature, a worse solution can be accepted with a higher probability, leading to an extensive search over the entire search space. As the temperature goes down, the probability to accept a worse solution decreases, and the algorithm gradually enters a refining phase. In our work, we integrate the Greedy Refinement with the SA to generate new solutions.

The GR algorithm is proposed in [14], which improves the Kernighan–Lin (KL) algorithm [17] to refine the bisection of a graph by iteratively swapping the pair of vertices that would most significantly reduce the edge cut until a local minimum is reached. The GR algorithm extends the KL algorithm so as to handle vertex weights, refines the multi-way partitioning and improves the running time.

For completeness, we present the GR algorithm as Algorithm 3. Given an existing partition of the virtual nodes, the nodes are checked in a random order. Consider node v in cluster l. Denote $ED[v]_h$ as the total traffic between v and its neighbors that belong to cluster h (where we allow $h = l$). The algorithm moves vertex v to the cluster with the highest value $ED[v]_h$ (or keeps it in the same cluster if it happens that $h = l$).

We now integrate this GR algorithm within a new point generation phase of SA: each time we randomly move a small number of nodes n_{exc} from one cluster l to another h, such that (1) the

Algorithm 3 Greedy Refinement Algorithm

```
Input:
    $k$: Number of clusters
    Initial assignment of VNodes to clusters
    $Rm_{cap}$: Remaining capacity of each cluster
    $Cap$: Maximum capacity for each cluster
    $Res$: Resource requirement vector, remaining capacity
Output: Final assignment of VNodes to clusters
 1: for $v \leftarrow$ random permutation from 1 to $n$ do
 2:     assume node $v \in$ cluster $l$
 3:     $ED[v]_h|_{h=1,\dots,k} \leftarrow 0$
 4:     for $u \leftarrow 1$ to $n$ do
 5:         if node $u \in$ cluster $h$ then
 6:             $ED[v]_h = ED[v]_h + w_{uv}$
 7:         end if
 8:     end for
 9:     $h = argmax\{ED[v]_h$ s.t. $W_i[v] + R^h_{cap} < Cap_h\}$
10:     move $v$ from cluster $l$ to $h$
11:     update $Rm^h_{Cap}$
12: end for
```

node that is moved from l to h should be on the "brink" (i.e., it should have at least one neighbor in the new cluster h), and (2) this movement does not violate the capacity constraints of cluster h. The exchange aims to introduce perturbation to the current solution so as to escape local minima. The procedure to generate new points is detailed in Algorithm 4. After the exchange is completed,

Algorithm 4 New Point Generation

```
Input:
    $k$: Number of clusters
    $n_{exc}$: Number of nodes to exchange
    Initial assignment of VNodes to clusters
    $Res = res_1, res_2, .., res_n$: Resource requirement vector
    $Cap_h = cap_1, cap_2, ..., cap_k$: Maximum capacity
Output: $P$: Final assignment of VNodes to clusters
 1: for $v \leftarrow$ random permutation from 1 to $n_{exc}$ do
 2:     node $v \in$ cluster $l$
 3:     if node $v$ has neighbor $\in$ cluster $h$ and
        $Cap_h + Req[v] < Cap_h$ then
 4:         move node $v$ from cluster $l$ to $h$.
 5:     end if
 6: end for
 7: for $t \leftarrow 1$ to $t_{ref}$ do
 8:     refine the partitioning via Greedy Refinement
 9: end for
```

we execute several iterations of the Greedy Refinement algorithm to refine the result.

The GR algorithm constructs a solution that represents a local optimum. The total outgoing weight of this solution is considered as the energy function for the SA algorithm. The SA algorithm will decide whether to accept this point or not. Since the solution passed to SA is already a local optimum point obtained by the GR algorithm, the SA moves around the local optimum points to find the final solution. This operation is more efficient than the naive implementation of randomly generating new points and letting the SA algorithm decide which solution to take.

Running time: The overall algorithm (Algorithm 1) consists of three steps, namely, computing the eigenvectors of the graph Laplacian, Constrained k-means, and graph refinement. The computation of the eigenvectors can be completed in $O(n^3)$ time. For the Constrained k-means, the clustering assignment subproblem can be solved in $O(kn \log n)$ time, where k is the number of clusters, and the cluster update problem can be solved in $O(kn)$ time. Let t_c be the number of iterations for Constrained k-means to converge; then, the total running time for the Constrained k-means is $O(kt_c nlgn)$. Using an adjacency table, each iteration of the refinement phase can be completed in time $O(E)$, where E is the total number of edges in a graph. If t_r iterations are needed, the refinement phase takes time $O(t_r E)$. Overall, this algorithm runs in $O(n^3 + kt_c n \log n + t_r E)$ time.

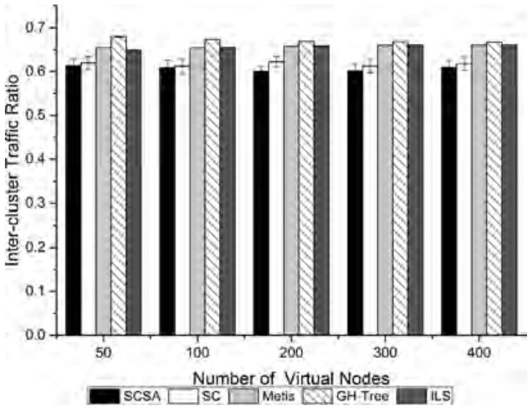

Fig. 4. Inter-cluster traffic ratio for k=3.

4. Experiments and evaluation

We now present the results of experiments we have conducted to evaluate the performance of spectral clustering (SC)-based algorithms. We use two methods to refine the partitioning: solely based on the GR algorithm (referred to as SC) as well as the SA-based refinement approach (SC-SA) we described in the previous section. We compare the results to those obtained using a Gomory-Hu tree [9], the METIS [15], and the ILS method [10].

4.1. Simulation setup

To evaluate the performance of our approach, we generate a traffic matrix as follows: each virtual node is connected to all other virtual nodes, and each node and link is assigned a weight to represent the resource and traffic requirements, respectively, and we also specify the maximum available capacity for each cluster. For each randomly generated traffic matrix, the vertex weight is uniformly distributed in (1, 2), the edge weight is uniformly distributed in (5, 20). Our goal is to partition the network into $k = 3, 4, 7$ clusters. For each cluster, we set the maximal capacity to 105% of the average weight of each cluster, i.e., the weight we place onto each cluster if we can have a perfect load balancing of the nodes. For the ILS algorithm, we set the number of iterations to be 5×10^4 and for each iteration, exchange 40% of nodes in each cluster. We also set $n_{exc} = 15$ for the new point generation phase in SA, and we run 3 iterations of the GR-algorithm to refine the partitioning result. We use two performance metrics: the inter-cluster traffic ratio (ITR), i.e., the ratio of the inter-cluster traffic to the total amount of traffic, and the running time of each algorithms. We run the simulation 50 times, and calculate the mean and 95% confidence intervals for the results.

4.2. Simulation results

Figs. 4and 5 plot the ITR and running time, respectively, against the number of virtual nodes and for $k = 3$ clusters. For the SA algorithm, we set the initial temperature to $T = 10^4$ and the maximum number of iterations to 600. We observe that spectral clustering

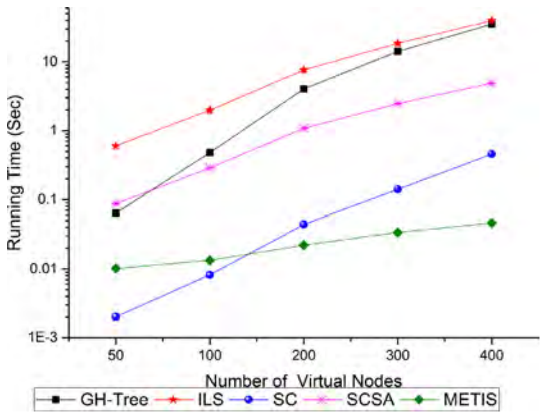

Fig. 5. Running time for k=3.

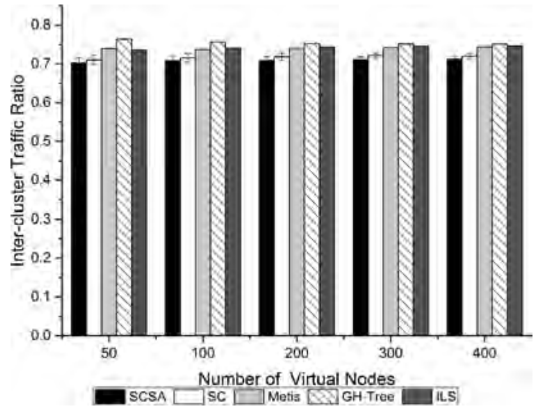

Fig. 6. Inter-cluster traffic ratio for k=4.

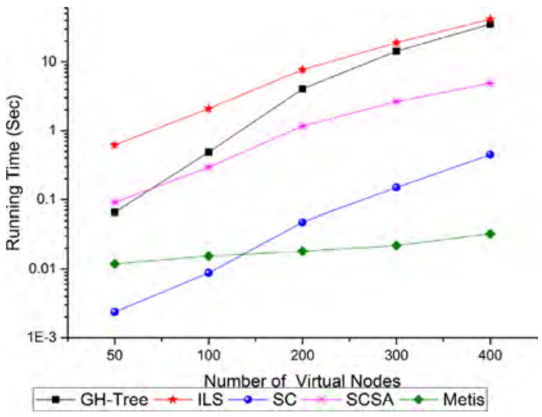

Fig. 7. Running time for k=4.

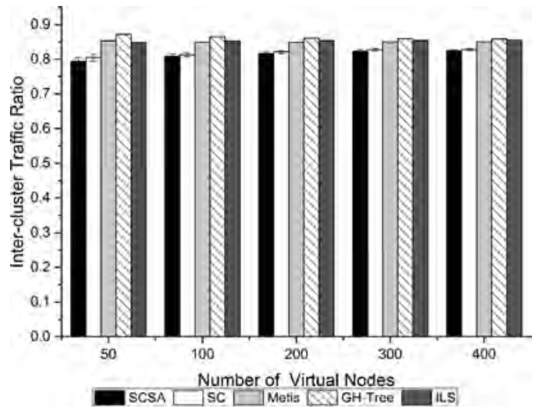

Fig. 8. Inter-cluster traffic ratio for k=7.

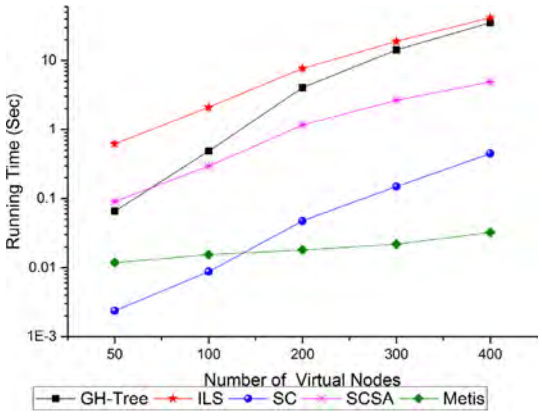

Fig. 9. Running time for k=7.

with SC-SA method is strictly better than the other algorithms in terms of inter-cluster traffic minimization. Compared with METIS, it reduces the inter-cluster traffic by 6–9% percent, with an average improvement of 8.3%. Compared with Gomory-Hu tree (respectively, ILS), inter-cluster traffic is reduced by as much as 11% (respectively, 9.7%), with an average improvement of 10% (respectively, 8.2%). Also, compared with the SC only, SC-SA based refinement produces an improvement of 1.6% on average. In terms of running time, with a small number of virtual machines, the SC algorithm has the similar performance with METIS while the running time for SC-SA stays close to the Gomory-Hu tree method, about one magnitude larger than the two above, while ILS takes still one magnitude longer. When there are more virtual nodes, we see that the GH-Tree has a similar performance with the ILS, about an order of magnitude larger than the SC-SA, while SC takes less running time by an order of magnitude than SC-SA, and METIS takes yet another order of magnitude less than SC.

The second set of simulation experiments is to partition the virtual request into $k = 4$ clusters, and the results are shown in Figs. 6 and 7. We kept the initial temperature for SA to $T = 10^4$ and the maximum number of iterations as 600. The SC algorithm produces clustering solutions that, in terms of inter-cluster traffic, outperform those produced by the METIS, Gomory-Hu tree, and ILS schemes by 4.5%, 6.5%, and 4.7%, respectively, on average. The SC-SA algorithm further reduces inter-cluster traffic by 1.1% on average, compared to SC. The running time results are similar to the experiments with $k = 3$ above.

Finally, Figs. 8 and 9 plot the results of the third set of simulation experiments where we set $k = 7$. The initial temperature for SA was set to $T = 10^5$ and the maximum number of iterations to 600. The results are similar to those of the first two experiments, in that, on average, the SC algorithm performs 4.5% better than METIS, 6.1% better than Gomory-Hu tree, and 4.8% better than ILS. Also, compared with SC, the SC-SA algorithm reduces inter-cluster traffic by a further 0.7% on average. In terms of running time, the relative behavior of the five algorithms is also similar to the last two experiments.

From this set of simulations, we conclude that the spectral clustering method with SA refinement produces the best solutions in terms of minimizing the inter-cluster traffic. It also compares favorably to existing clustering approaches based on ILS and Gomory-Hu tree, in terms of running time. When we compare with METIS, we found out its performance is at trade-off with METIS: the spectral clustering based algorithm achieves a better performance in

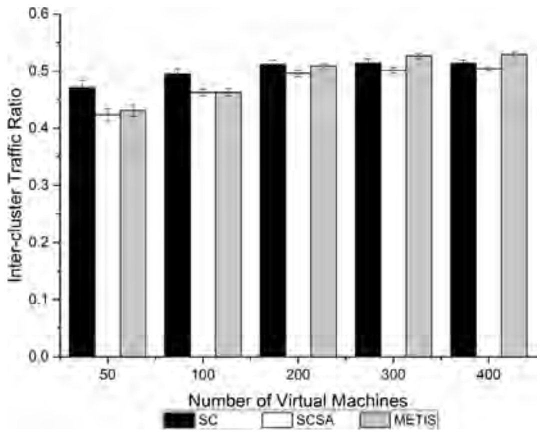

Fig. 10. Inter-cluster traffic ratio for modular pattern.

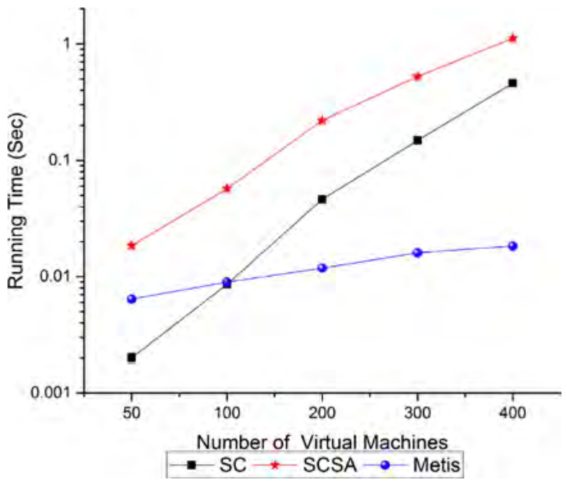

Fig. 11. Running time for modular pattern.

minimizing the inter-cluster traffic, while METIS achieves a lower execution time.

4.3. Further performance comparison with METIS

To further evaluate the performance of our proposed algorithm, we compare its performance with METIS on three different types of topologies with different partitioning settings. Among the entire set of simulations, we selectively present three groups of evaluation results that illustrate the relative performance of the algorithm.

First, we evaluate the result on random modular graphs [24]. This intends to capture the pattern that communication taking place within a group of VM is more intensive than that between groups of VMs. We assume the topology contains 20 clusters of equal size, and the probability of connecting one pair of nodes within the same cluster is 80%, while connecting one pair of nodes from two different clusters is 20%. We generate the topology using this model, and the traffic on each link follows a Gaussian distribution, with a mean of 100 and variance of 25. We partition the generated graph into 3 clusters, and the capacity constraint is set to 120% of the average. This setting for the capacity constraint tries to capture the condition that abundant computing resources in the underlying network are provided for each cluster such that a more flexible partitioning of the virtual request is allowed. The results are presented in the Fig. 10.

In this set of simulations, we can see that inter-cluster traffic is minimized by SC-SA with an additional 2.7% on average when compares with METIS. However, when we compare the result between METIS and SC, we can see that METIS achieves a better result on smaller requests by as much as 9%, while SC achieves a better result on larger requests by around 3%. On average, METIS brings about 3% of improvement than SC for traffic minimization.

Next, we generate the topology following the Waxman model [23]. The Waxman model has been used to model the Internet, especially the intra-domain networks [22]. In a Waxman model, nodes are randomly placed on a rectangle area, and the connectivity probability between a pair of nodes is based on their Euclidean distance, more formally, $p = \alpha \exp(-d/(\beta * L))$, where d is the Euclidean distance between one pair of nodes, and L is the maximum distance allowed. The α and β are two parameters that define the connectivity pattern. More specifically, α defines the overall connectivity probability (a higher α results in a denser connectivity) while β restricts the probability of connection based on the distance, i.e., a larger β tends to increase the chance of connection for a pair of nodes that are far from each other, while a lower β tends to prevent such connection. In this experiment, we use $L = 100$, $\alpha = 0.05$, and $\beta = 0.3$ to define the connectivity of the virtual request. For the partitioning setting, we divide the virtual request into 4 clusters with different capacities, each cluster with 20%, 20%, 30%, 30% of the total weight of the request. Maximum violation for the capacity is defined to be 5%. This setting represents a situation where the distribution of the computational resource is imbalanced, such as heterogeneous resources, or the workload is imbalanced across different domains. The simulation setting is shown in Fig. 12.

As we can see from the figures, SC-SA achieves an improvement of 5% compared with METIS for ITR minimization, while SC delivers a 2.5% improvement compared with METIS.

Finally, we evaluate this algorithm on a virtual request whose degree distribution exhibits power law, which characterizes the degree distribution of router-level and AS-level Internet graphs [21]. We use the Barabasi-Albert (BA) model [20] to obtain a traffic pattern that follows a power-law. It starts with a small connected component, and then probabilistically attaches a new vertex to the existing vertex following preferential attachment, i.e., the likelihood of connection depends on the degree of the existing vertex. We start with 5 connected vertices and end up with a topology consisting of 50 to 400 vertices. We partition the graph into four clusters of equal capacity, while allowing a cluster to exceed its capacity limit by 5%. The simulation results are shown in Fig. 14.

Compared with METIS, the simulation result suggests an average improvement of 34% in terms of ITR minimization for SC-SA, with a maximum improvement of 43%, while SC brings forth an improvement of 14% compared with METIS on average.

The running time for the three different types of topologies is similar to the previous set of simulations., and it is shown in Figs. 11, 13 and 15. METIS requires less execution time compared with SC-SA, by two orders of magnitude faster in general. Compared with SC, we see that their performance is similar on a request with small scale, while on a larger request, METIS will still work better.

4.4. Detailed comparison of SC To METIS

First, we observe that while our algorithm can achieve a better result in minimizing the inter-cluster traffic, the performance of our proposed algorithm varies across different traffic patterns. Indeed, compared with the performance on other traffic models, the SC-based algorithm reduces a higher amount of inter-cluster traffic on the BA model. Such a difference results from the fact that the degree distribution with the BA model follows a power-law, which implies a highly irregular graph. This means that some network

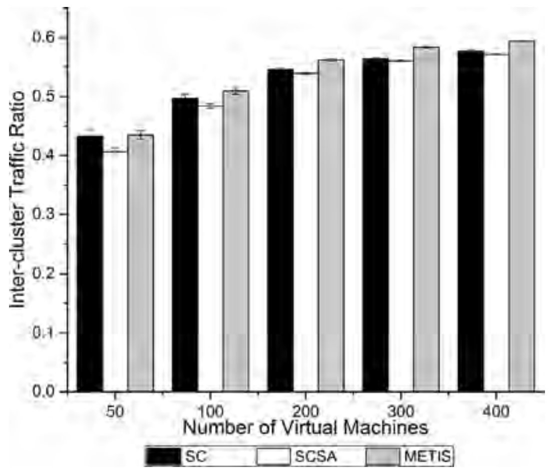

Fig. 12. Inter-cluster traffic ratio for Waxman model pattern.

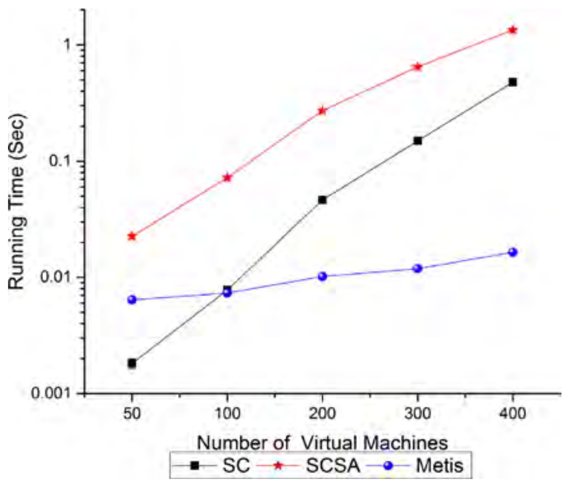

Fig. 13. Running time for Waxman model pattern.

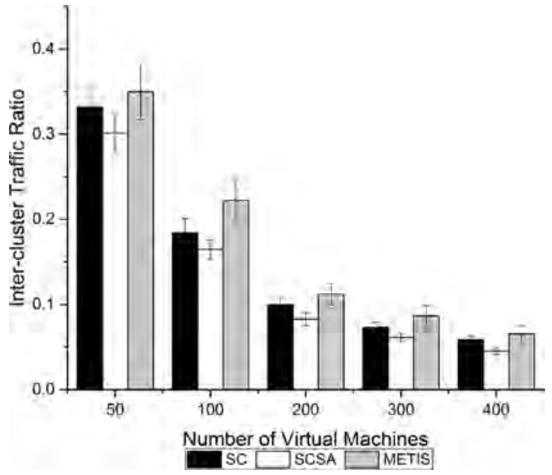

Fig. 14. Inter-cluster traffic ratio for BA-model pattern.

nodes may be richly connected, and the traffic flow in and out of those nodes is significantly higher than the rest.

Under this scenario, METIS, with its greedy partitioning growing scheme, may fail to obtain a high-quality solution. It will first randomly select a node as the starting point, and add it to the growing partition. The gain of neighboring vertices of this partition, defined as the traffic reduction in the inclusion of one particular vertex, will then be updated. The nodes with the largest gain will be added to the partition. This process will repeat until the desired size of the partition is achieved.

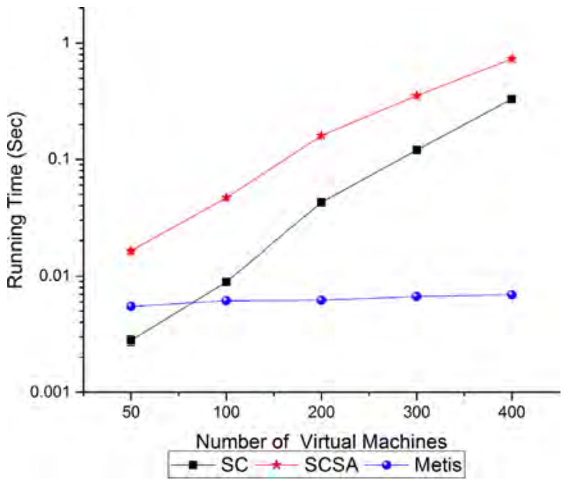

Fig. 15. Running time for BA-model pattern.

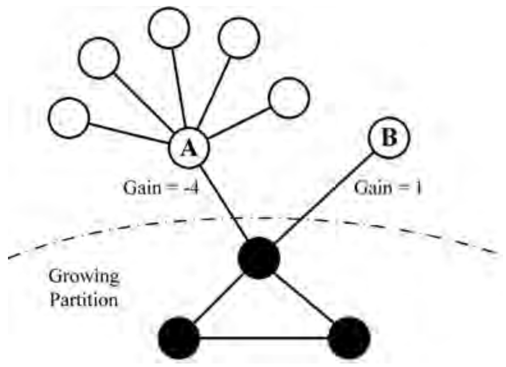

Fig. 16. A growing partition to two nodes.

When it comes to the BA model, we observe that the power-law distribution, with its “fat-tail” property, may contain a large amount of nodes with a small degree. Also, there will exist some nodes with a very large degree. When we randomly select a node as the starting point, given the population of the nodes with a small degree, it is likely that this node will be one of these small degree nodes. In the greedy spanning step, the nodes with a large degree, however, will likely be deferred to be added to the cluster. Take the example in Fig. 16, and let us assume unit traffic for each edge. When the growing partitioning reaches nodes *A* and *B* into the growing partition, the gains for the two nodes are -4 and 1 respectively, which leads to inclusion of *B*. This shows that when we are computing the gain of a node associated with heavy traffic, if many of its neighbors are not included within the growing partition, we shall always end up with a large negative gain. This large negative gain will result in the deferral of adding of this node to the growing partition.

Since the assignment of nodes with heavy traffic may turn out to be the key to a good partitioning result, the deferral in considering those nodes will lead to a sub-optimal performance. An illustrating example to this can be found in Fig 17. Fig. 17 (a) shows the underlying connectivity of a network request. Suppose all the traffic and nodes are of equal weight, and the objective is to obtain a bisection of the requests. In this case, it is easy to verify that an optimal solution would be to put nodes $a-g$ in one cluster, and nodes $h-n$ in another. However, if we are applying the greedy growing partitioning approach, with the starting point being *e*, one

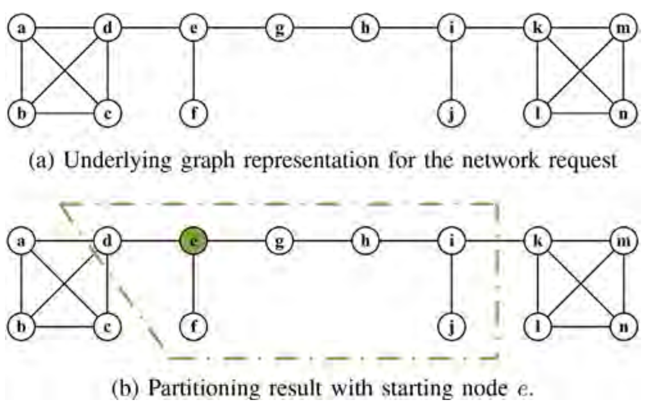
(a) Underlying graph representation for the network request
(b) Partitioning result with starting node e.

Fig. 17. Example of the outcome on the partitioning.

Table 1
First 2 smallest eigenvectors.

Node	a	b	c	d	e	f	g
evec 1	0.26	0.26	0.26	0.26	0.26	0.26	0.26
evec 2	−0.32	−0.32	−0.32	−0.30	−0.22	−0.20	−0.07
node	h	i	j	k	l	m	n
evec 1	0.26	0.26	0.26	0.26	0.26	0.26	0.26
evec 2	0.07	0.20	0.22	0.30	0.32	0.32	0.32

possible sequence of adding nodes to the cluster would be *e, f, g, h, i, j, d*, resulting in a partitioning shown as in Fig. 17(b). The reason for producing this suboptimal solution is that, the growing partition will defer adding vertex *d* or *k* into the cluster, and thus eliminate the possibility of achieving an optimal solution.

The SC based approach, with its ability to achieve a relaxed solution to the global optimal solution, on the other hand, can achieve a better performance. Table 1 presents the first two smallest eigenvectors (denoted as evec 1 and evec 2, respectively). Observe that all the elements in the first smallest eigenvector remain to be same, in fact this will hold for all the traffic matrix. As we look at data points featured by the eigenvectors, we can verify that Euclidean distances for the connected nodes are smaller in the eigenspace, unaffected by the degree distribution, which leads to a better performance in this case.

Second, we observe that it will take a longer time for the SC-based algorithm to reach a solution. Following from our discussion in Section 3, the time complexity for SC-based algorithm would be of $O(n^3 + kt_c n\log n + t_r E)$. As for the METIS, with the graph coarsening algorithm to reduce the problem size, the partitioning result can be achieved in $O(E)$ time. Notably, for a traffic matrix with a moderate size (e.g. up to 100 nodes), the amount of time the SC takes is smaller than METIS, while for a request consisting of 400 nodes, the partitioning can be done by our algorithm in about 1 s. This implies that the setup delay caused by our algorithm would be acceptable in most cases.

Overall, we conclude that our proposed algorithm can consistently delivery a better performance in terms of traffic minimization, and the results are more robust when the amount of traffic significantly varies across different network nodes. METIS can achieve a better running time, especially when the scale of the network is large.

5. Conclusion

Resource allocation with respect to the network traffic is essential to the scalability and stability of network virtualization. To this end, we designed a network-aware virtual request partitioning scheme that produces clusters with low inter-cluster traffic. We use a Constrained *k*-means algorithm in the clustering phase of spectral clustering to ensure that cluster capacity constraints are met. Also, we have developed an algorithm based on simulating annealing to efficiently refine the resulting clustering solution. Our algorithm constructs high-quality solutions within a reasonable amount of time and compares favorably to existing approaches.

References

[1] A. Wang, M. Iyer, R. Dutta, G.N. Rouskas, I. Baldine, Network virtualization: Technologies, perspectives, and frontiers, J. Lightwave Technol. 31 (4) (2013) 523–537.
[2] N. Chowdhury, R. Boutaba, Network virtualization: state of the art and research challenges, Commun. Mag., IEEE 47 (7) (2009) 20–26.
[3] M.F. Bari, R. Boutaba, R. Esteves, L.Z. Granville, M. Podlesny, M.G. Rabbani, Q. Zhang, M.F. Zhani, Data center network virtualization: a survey, Commun. Surv. Tut., IEEE 15 (2) (2013) 909–928.
[4] A. Fischer, J.F. Botero, M.T. Beck, H. De Meer, X. Hesselbach, Virtual network embedding: a survey, Commun. Surv. Tut., IEEE 15 (4) (2013) 1888–1906.
[5] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, E. Silvera, A stable network-aware vm placement for cloud systems, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), IEEE Computer Society, 2012, pp. 498–506.
[6] Y. Xin, I. Baldine, A. Mandal, C. Heermann, J. Chase, A. Yumerefendi, Embedding virtual topologies in networked clouds, in: Proceedings of the 6th International Conference on Future Internet Technologies, ACM, 2011, pp. 26–29.
[7] I. Houidi, W. Louati, W.B. Ameur, D. Zeghlache, Virtual network provisioning across multiple substrate networks, Comput. Networks 55 (4) (2011) 1011–1023.
[8] P.T. Endo, A.V. de Almeida Palhares, N.N. Pereira, G.E. Goncalves, D. Sadok, J. Kelner, B. Melander, J.E. Mångs, Resource allocation for distributed cloud: concepts and research challenges, Network, IEEE 25 (4) (2011) 42–46.
[9] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: INFOCOM, 2010 Proceedings IEEE, IEEE, 2010, pp. 1–9.
[10] A. Leivadeas, C. Papagianni, S. Papavassiliou, Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning, Parallel Distrib. Syst., IEEE Trans. 24 (6) (2013) 1077–1086.
[11] U.V. Luxburg, A tutorial on spectral clustering, Stat. Comput. 17 (4) (2007) 395–416.
[12] P. Bradley, K. Bennett, A. Demiriz, Constrained k-Means Clustering, Microsoft Research, Redmond, 2000, pp. 1–8.
[13] P.S. Bradley, O.L. Mangasarian, W.N. Street, Clustering via concave minimization, Adv. Neural Inf. Process. Syst. (1997) 368–374.
[14] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Scient. Comput. 20 (1) (1998) 359–392.
[15] G. Karypis, V. Kumar, Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0, 1995.
[16] K. Andreev, H. Racke, Balanced graph partitioning, Theory Comput. Syst. 39 (6) (2006) 929–939.
[17] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (2) (1970) 291–307.
[18] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, Volume 1, Oakland, CA, USA., 1967, pp. 281–297.
[19] A.Y. Ng, M.I. Jordan, Y. Weiss, On spectral clustering: analysis and an algorithm, Adv. Neural Inf. Process. Syst. 2 (2002) 849–856.
[20] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[21] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the internet topology, in: ACM SIGCOMM computer communication review, volume 29, ACM, 1999, pp. 251–262.
[22] M. Naldi, Connectivity of waxman topology models, Comput. Commun. 29 (1) (2005) 24–31.
[23] B.M. Waxman, Routing of multipoint connections, IEEE J. Sel. Areas Commun. 6 (9) (1988) 1617–1622.
[24] L. Huang, K. Park, Y.-C. Lai, Information propagation on modular networks, Phys. Rev. E 73 (3) (2006) 035103.
[25] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, C. Schulz, Recent advances in graph partitioning, in: Algorithm Engineering, Springer, 2016, pp. 117–158.
[26] B. Hendrickson, R. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations, SIAM J. Scient. Comput. 16 (2) (1995) 452–469.
[27] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, et al., Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

Lingnan Gao is currently a Ph.D. candidate in Department of Computer Science, North Carolina State University. He received his B.Eng. degree in Communication Engineering from Beijing University of Posts and Telecommunications in 2014. His research interest includes network virtualization, network design and optimization.

George N. Rouskas Professor and Director of Graduate Programs in the Department of Computer Science at NC State University. He received a BS from the National Technical University of Athens, Greece, and the MS and PhD degrees from the Georgia Institute of Technology. His research contributions have been in the areas of network architectures, optical networks, network design and optimization, and performance evaluation. He is an IEEE Fellow, the Chair of ONTC, the Chair of the IEEE Comsoc Distinguished Lecturer Selection Committee, and was a Distinguished Lecturer for IEEE Comsoc in 2010-2011. He received an NSF CAREER Award, the 2004 ALCOA Foundation Engineering Research Achievement Award and the 2003 NC State Alumni Outstanding Research Award, and he was inducted in the NC State Academy of Outstanding Teachers in 2004. He is the founding Editor-in-Chief of the Optical Switching and Networking Journal, and he has served on the Editorial Boards of IEEE/ACM Transactions on Networking, IEEE/OSA JOCN, and IEEE/OSA JLT. He was Chair or co-Chair of numerous conferences including IEEE ICNP 2014, IEEE GLOBECOM 2010 ONS, IEEE LANMAN 2004 and 2005, and currently co-chairs IEEE ICC 2017 ONS. He has served on Best Paper Award Committees for several conferences, including IEEE INFOCOM 2011 and IEEE ICC 2016. Within ONTC, he has worked to bring together the optical networking community, enhance membership value, and further growth by embracing emerging technology directions.