Network-Aware Virtual Request Partitioning Based on Spectral Clustering

Lingnan Gao, George N. Rouskas
North Carolina State University, Raleigh, NC 27695-8206 USA

***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 an optimal 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.**

I. Introduction

Network virtualization is seen as crucial 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 the end users services 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].

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

In either area, mapping virtual 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 II. In Section III, we formally define the problem and present the various components of the virtual request partitioning algorithm. In Section IV, we present the results of simulation experiments we have conducted to compare the performance of this algorithm to existing algorithms. We conclude the paper in Section V.

II. 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. One shortcoming of this method is that the resulting clusters may be highly imbalanced in terms of the number of nodes they contain.

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.

III. 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. Figure III(a) shows a set of virtual nodes that have been partitioned in three clusters such that traffic between clusters is minimum. In the VNE scenario of Figure III(b)), mapping each of the clusters to a different domain will minimize onter-domain traffic (which presumably is more expensive than intra-domain traffic). In the context of the VMP problem in Figure III(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 partition problem as it applies to both the VNE and VMP probleme, and then present an algorithm based on spectral clustering.

A. Problem Statement

We model the communication between virtual nodes as a traffic matrix $W = [w_{ij}]$, where element w_{ij} represents the data rate 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 interger linear program (ILP):

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

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

$$\sum_j y_{ij}^k = x_i^k, \qquad \forall i, k \tag{3}$$

$$\sum_k x_i^k = 1, \qquad \forall i \tag{4}$$

$$x_i^k = \{0,1\}, y_{ij}^k = \{0,1\} \tag{5}$$

The binary variable $x_i^k \in \{0,1\}$ here indicates if virtual node i is assigned to cluster k while binary variable $y_{ij}^k \in \{0,1\}$: binary variables 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) guarantees consistency between decision variable x and y. Constraint (4) 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 [17]. We also note that by replacing "virtual node" with "VM" and cluster with "processor," the above formulation also expresses the problem of placing VMs onto processors so as to minimize inter-processor traffic.

We apply a spectral clustering [12] approach to solving the virtual request partitioning problem. The general procedure is outlined as Algorithm 1 below, 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
$R = r_1, r_2, ..., r_n$: resource requirement of VNodes
$Cap = cap_1, cap_2, ..., cap_k$: 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 the unnormalized Laplacian $L = (D - W)$
3: Solve the generalized eigenproblem $Lu = \lambda Du$
4: Obtain the eigenvector associated with the k smallest eigenvalues
5: Let matrix U contain the above eigenvectors as columns
6: Let z_i be the vector associated with i^{th} row of U.
7: Cluster the points $(z_i)|_{i=1...n}$ under the capacity constraints Cap via **Constrained-K-means**
8: Refine the partitioning result by **Greedy-Refinement** based Simulated Annealing

B. Spectral Clustering

Spectral clustering [12] is used to find a set of clusters such that the edges between clusters have low weights (in this case, the weights represent inter-cluster traffic). An important feature of spectral clustering is that, unlike the conventiona 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 an $n \times n$ traffic matrix W, the normalized Laplacian matrix is defined as $L_{rw} = D^{-1}(D - W)$, 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 $NCut$ metric as:

$$NCut(P_1, P_2, ..., P_n) = \sum_{i=1}^{k} \frac{cut(P_i, \bar{P}_i)}{vol(P_i)} \tag{6}$$

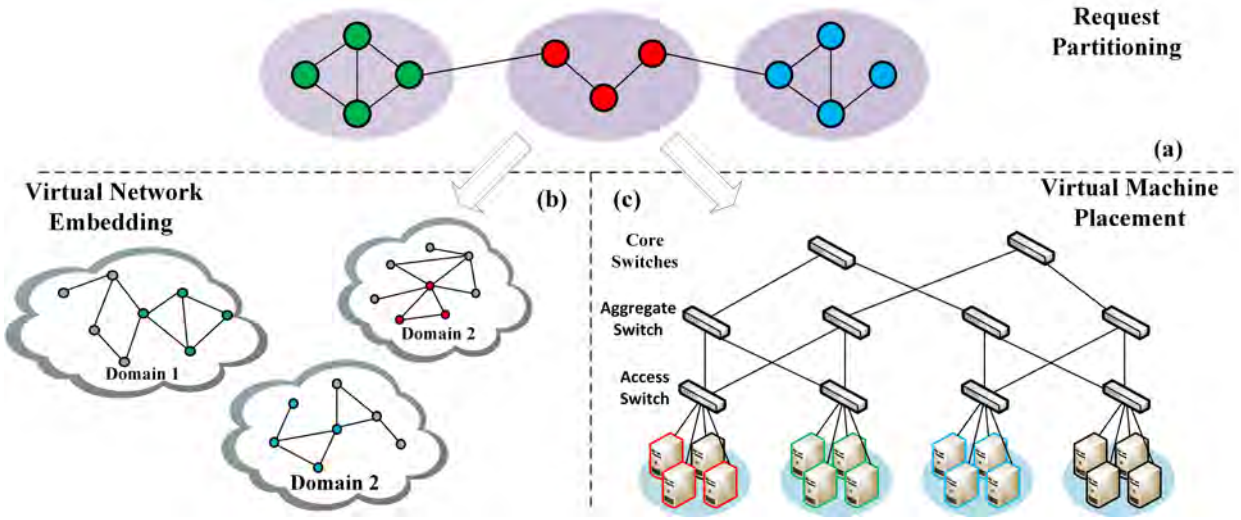

Fig. 1. Virtual request paritioning for Virtual Network Embedding and Virtual Machine Placement

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

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

$$h_{ij} = \begin{cases} 1/\sqrt{vol(P_i)} & if v_i \in P_j \\ 0 & otherwise \end{cases} \quad (7)$$

then, the trace of the product $H^T LH$ can be written as $Tr(H^T LH) = \sum_{i=1}^{n} \frac{cut(P_i, \bar{P}_i)}{|vol(P_i)|} = NCut(P_1, P_2, ..., P_n)$. Also observe that $H^T H = I$. Therefore, we can reformulate the problem of minimizing $NCut$ as follows:

$$\begin{aligned} \textbf{minimize} \quad & H^T LH \\ \textbf{subject to} \quad & H^T H = I \\ & h_{ij} = \{1/\sqrt{vol(P_i)}, 0\} \end{aligned} \quad (8)$$

We can obtain an approximate solution to this problem in polynomial time by relaxing the last condition. According to the Rayleigh-Ritz Theorem, the solution to the relaxed problem would be to take H as the k smallest eigenvectors of L_{rw}, i.e. the eigenvectors corresponding to the k smallest eigenvalues. For the proof of this property and other details relating to spectral clustering, please refer to [12].

Let matrix U be a $n \times k$ matrix that contains the above k eigenvectors as columns, and let z_i be the vector associated with the i-th row of U. To obtain the final solution, a clustering algorithm may be employed 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.

C. Constrained K-means

Conventional spectral clustering uses the k-means algorithm [19] 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 [13]. 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 [13] is:

$$\begin{aligned} \textbf{minimize} \quad & \sum_{i=1}^{m} \sum_{h=1}^{k} S_{ih} \cdot \left(\frac{1}{2} \| z^i - C_h \|_2^2 \right) \\ \textbf{subject to} \quad & \sum_{h=1}^{k} S_{ih} = 1, \ \forall i; \quad S_{ih} \geq 0, \ \forall i, \ \forall h. \\ & \sum_{i} S_{ih} \geq \tau_h \forall h \end{aligned} \quad (9)$$

In this formulation, S_{ih} is a selection variable denoting wheter 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 τ_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 $\sum_i S_{ih} \geq \tau_h$ with $\sum_i r_i S_{ih} \leq Cap_h$, and follow the iterative method proposed in [13].

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.

$$\begin{aligned} \textbf{minimize} \quad & \sum_{i=1}^{m}\sum_{h=1}^{k} S_{ih}\cdot\left(\frac{1}{2}\|z^i - C_h\|_2^2\right) \\ \textbf{subject to} \quad & \sum_{h=1}^{k} S_{ih} = 1, && \forall i \\ & \sum_{i} r_i S_{ih} \le Cap_h, && \forall h \\ & S_{ih} \ge 0,\ \forall i, && \forall h \end{aligned} \tag{10}$$

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

$$C_h^{t+1} = \begin{cases} \frac{\sum_{i=1}^{m} S_{ih}^t x^i}{\sum_{i=1}^{m} S_{ih}^t} & if \sum_{i=1}^{m} S_{ih}^t > 0 \\ C_i^t & otherwise \end{cases} \tag{11}$$

It was shown in [14] that cluster assignment is 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 [14]. 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 $(z_i)|_{i=1,...,n}$, 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 $MaximumCapacity$ 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 infinity. An example of the reprsentation of a cluster assignment problem to an MCF network is shown in Figure III-C(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 Figure III-C(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 ot 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_{cap}^h, and the remaining resource requirement of each virtual node i as Rm_{res}^i. Our constrained$-k$-means with greedy cluster assignment algorithm is shown as Algorithm 2.

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 S_{ih}

Iteration $t \leftarrow 0$, randomly initialize $C_h^t\ \forall h$
Remaining resource requirement $Rm_{req} \leftarrow R$
Remaining capacity $Rm_{cap} \leftarrow Cap$
while $C^{t+1} \neq C^t$ **do**
  Compute pairwise distance between data points and cluster centers $D = \{d_{11}, d_{12}, ..., d_{nk}\}$
  Ascending sort D to get $D_{asc} = \{d'_1, d'_2, ..., d'_{n*k}\}$
  Clustering Assignment:
  for $j \leftarrow 1\ to\ (n*k)$ **do**
    Choose point i and center point h associated with d'_j
    if $Rm_{req}^i < Rm_{cap}^h$ **then**
      $S_{ih} \leftarrow Rm_{req}^i / r_i$; $Rm_{cap}^h \leftarrow Rm_{cap}^h - R_{res}^i$; $Rm_{res}^i \leftarrow 0$
    else
      $S_{ih} \leftarrow Rm_{cap}^h / r_i$; $Rm_{req}^i \leftarrow Rm_{req}^i - Rm_{cap}^h$; $Rm_{cap}^h \leftarrow 0$
    end if
  end for
  Clustering Update:
  update the center points according to (11)
  $t \leftarrow t+1$
end while
$P \leftarrow$ round S without violating capacity constraint.

Note that the 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 S_{ih} 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 data points will get fragmented on that cluster, and therefore, the number of fractional data points will be less than the number 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.

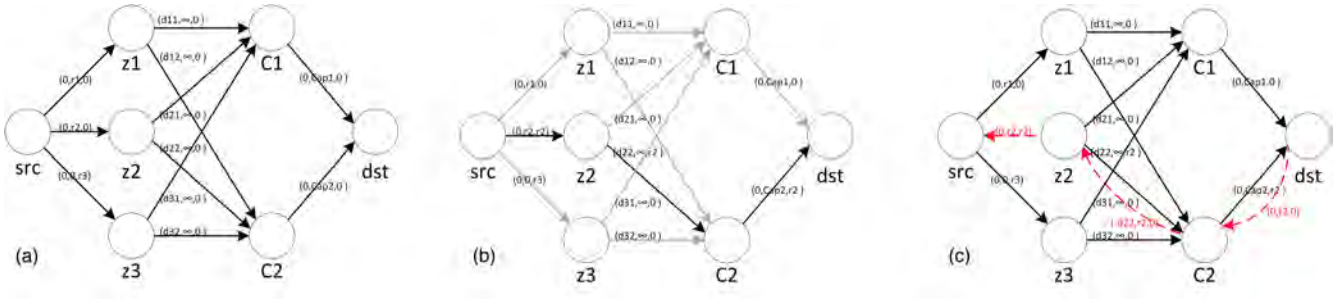

Fig. 2. MCF view of clustering assignment subproblem

D. Partitioning Refinement

In order to improve upon the partition obtained by the Constrained-k-means algorithm, we employ a greedy refinement (GR) method that employs simulated annealing (SA). The GR algorithm is inspired by an algorithm proposed in [15], which improves the Kernighan-Lin (KL) algorithm [18] 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.

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

for $v \leftarrow$ random permutation from 1 *to* n **do**
  assume node $v \in$ cluster l
  $ED[v]_h|_{h=1,...,k} \leftarrow 0$
  for $u \leftarrow 1$ *to* n **do**
    if node $u \in$ cluster h **then**
      $ED[v]_h = ED[v]_h + w_{uv}$
    end if
  end for
  h = $argmax\{ED[v]_h$ s.t. $W_i[v] + R^h_{cap} < Cap_h\}$
  move v from cluster l to h
  update Rm^h_{Cap}
end for

For completeness, we present the GR algorithm as Algorithm 3. Given a partition of the virtual nodes into clusters, 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 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 solusion so as to escape local minima. The procedure to generate new points is detailed in Algorithm 4. After the exchange is conpleted, 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 lettting the SA algorithm decide which solution to take.

Algorithm 4 New Point Generation

Input:
k: Number of clusters
n_{exc}: Number of nodes to exchange
Initial assignment of VNodes to clusters
$Res = res_1, res_2, .., p_n$: Resource requirement vector
$Cap_h = cap_1, cap_2, ..., cap_k$: Maximum capacity
Output: P: Final assignment of VNodes to clusters

for $v \leftarrow$ random permutation from 1 *to* n_{exc} **do**
  node $v \in$ cluster l
  if node v has neighbor $\in$ cluster h **and** $Cap_h + Req[v] < Cap_h$ **then**
    move node v from cluster l to h.
  end if
end for
for $t \leftarrow 1$ *to* t_{ref} **do**
  refine the partitioning via **Greedy Refinement**
end for

Running time: The overall algorithm (Algorithm 1) consist of three steps, namely, computing the eigenvectors of the graph

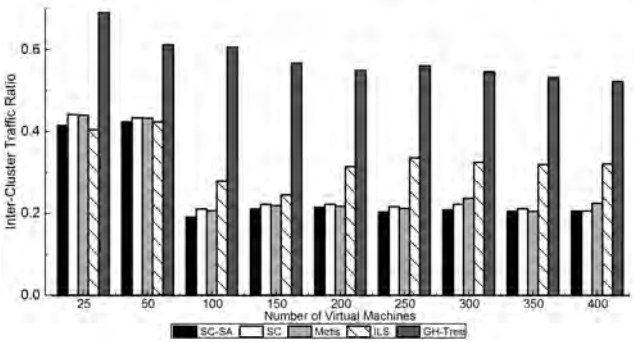

Fig. 3. Inter-cluster Traffic Ratio for k=3

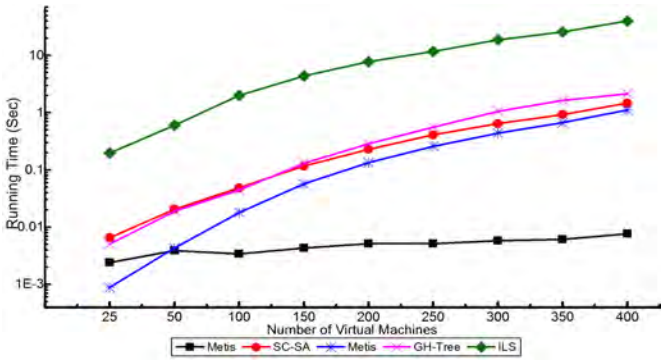

Fig. 4. Running Time for k=3

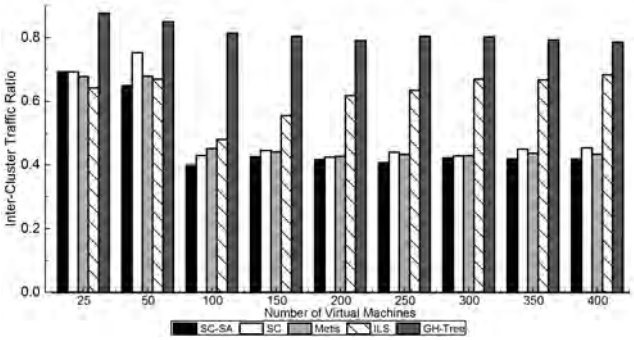

Fig. 5. Inter-cluster Traffic Ratio for k=4

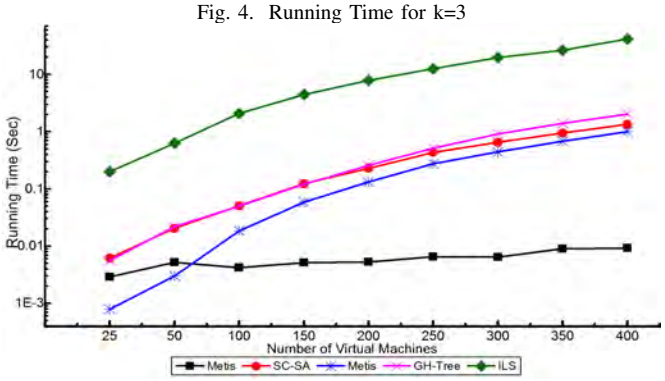

Fig. 6. Running Time for k=4

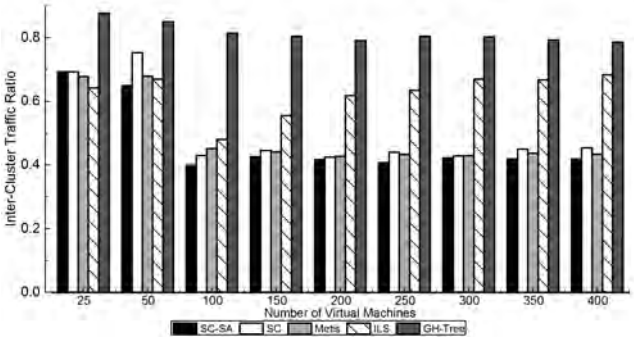

Fig. 7. Inter-cluster Traffic Ratio for k=7

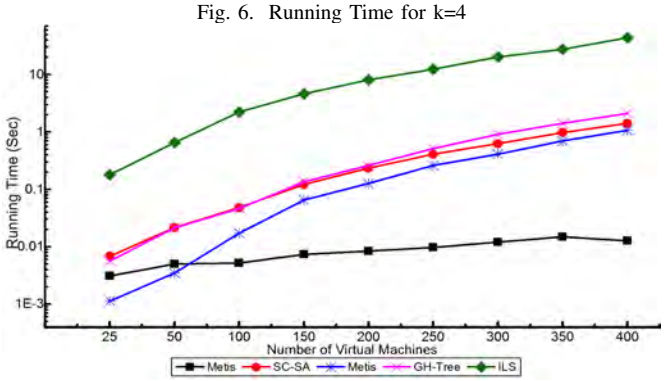

Fig. 8. Running Time for k=7.

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 $O(E)$. 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.

IV. Experiments and Evaluation

We now present the results of experiments we have conducted to evaluate the performance of spectral clustering (SC)-based Algorithms 1. 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 algorithm [16], and the ILS method [10].

We use randomly generated topologies to test the performance of these algorithms. Each node and link in the network 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 radomly generated topology, the vertex weight is uniformly distributed in $(1, 2)$, the edge weight is uniformly distributed in $(20, 60)$. Our goal is to partition the network into $k = 3, 4, 7$ clusters. For each cluster, we set the maximal capacity to 103% of the average.

For the ILS algorithm, we set the number of iterations to be 5×10^4. We also set $n_{exc} = 15$ for new point generation phase in SA. 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.

Figures 3 and 4 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 150. We observe that spectral clustering 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 1-12% percent, with an average improvement of 5.0%. Compared with Gomory-Hu tree (respectively, ILS), inter-cluster traffic is reduced by as much as 69% (respectively, 49%), with an average improvement of 57% (respectively, 24%). Also, compared with the SC only, SC-SA based refinement produces an improvement of 4.6% on average. In terms of running time, the SC and SC-SA algorithms perform similar to the Gomory-Hu tree method, taking just a few seconds for virtual requests with 400 nodes. The running time of METIS is significantly smaller, especially for large problem instances, while ILS takes about one order of magnitude longer than the three algorithms above.

The second set of simulation experiments is to partition the virtal request into $k = 4$ clusters, and the results are shown in Figures 5 and 6. We kept the initial temperature for SA to $T = 10^4$ and the maximum number of iterations as 150. 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 3.3%, 45%, and 34%, respectively, on avearage. The SC-SA algorithm further reduces inter-cluster traffic by 2.7% on average, compared to SC. The running time results are similar to the experiments with $k = 3$ above.

Finally, Figures 7 and 8 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 250. The results are similar to those of the first two experiments, in that, on average, the SC algorithm performs 3.5% better than METIS, 42% better than Gomory-Hu tree, and 24% better than ILS. Also, compared with SC, the SC-SA algorithm reduces inter-cluster traffic by a further 4% 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. We also note that, while METIS is faster than all clustering methods considered in our study, it partitions a graph into a set of components of roughly equal weight, while our proposed scheme is more general.

V. Conclusion

We have designed a network-aware virtual request partitioning scheme that produces clusters that minimize the 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] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[2] N. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, 2009.

[3] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 2, pp. 909–928, 2013.

[4] A. Fischer, J. F. Botero, M. Till Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 4, pp. 1888–1906, 2013.

[5] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and 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)*, pp. 498–506, IEEE Computer Society, 2012.

[6] Y. Xin, I. Baldine, A. Mandal, C. Heermann, J. Chase, and A. Yumerefendi, "Embedding virtual topologies in networked clouds," in *Proceedings of the 6th International Conference on Future Internet Technologies*, pp. 26–29, ACM, 2011.

[7] I. Houidi, W. Louati, W. B. Ameur, and D. Zeghlache, "Virtual network provisioning across multiple substrate networks," *Computer Networks*, vol. 55, no. 4, pp. 1011–1023, 2011.

[8] P. T. Endo, A. V. de Almeida Palhares, N. N. Pereira, G. E. Goncalves, D. Sadok, J. Kelner, B. Melander, and J.-E. Mångs, "Resource allocation for distributed cloud: concepts and research challenges," *Network, IEEE*, vol. 25, no. 4, pp. 42–46, 2011.

[9] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[10] A. Leivadeas, C. Papagianni, and S. Papavassiliou, "Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1077–1086, 2013.

[11] D. Wagner and F. Wagner, *Between min cut and graph bisection*. Springer, 1993.

[12] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[13] P. Bradley, K. Bennett, and A. Demiriz, "Constrained k-means clustering," *Microsoft Research, Redmond*, pp. 1–8, 2000.

[14] P. S. Bradley, O. L. Mangasarian, and W. N. Street, "Clustering via concave minimization," *Advances in neural information processing systems*, pp. 368–374, 1997.

[15] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[16] G. Karypis and V. Kumar, "Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.

[17] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems*, vol. 39, no. 6, pp. 929–939, 2006.

[18] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[19] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA., 1967.