An Algorithmic Approach to Datacenter Cabling

Rachit Agarwal, Jayaram Mudigonda, Praveen Yalagandula, Jeffrey C. Mogul

HP Laboratories
HPL-2015-8

Abstract:

Datacenter topology design is a complex problem with a huge search space. Recent work on systematic solution space exploration points out a significant problem on cabling: how to map a logical topology with servers, switches, and links onto a physical space with racks and cable trays such that the cable costs are minimized? In this paper, we show that this problem is NP-hard and present partitioning-based heuristics. Evaluation with different topologies demonstrate that our approach discovers better layouts than the previous approaches and reduces the cabling costs by up to 38%.

External Posting Date: February 6, 2015 [Fulltext] Approved for External Publication
Internal Posting Date: February 6, 2015 [Fulltext]

An Algorithmic Approach to Datacenter Cabling

Rachit Agarwal*, Jayaram Mudigonda†, Praveen Yalagandula‡, and Jeffrey C. Mogul†
* University of California, Berkeley
† Google, Inc.
‡ Avi Networks

***Abstract*—Datacenter topology design is a complex problem with a huge search space. Recent work on systematic solution space exploration points out a significant problem on cabling: how to map a logical topology with servers, switches, and links on to a physical space with racks and cable trays such that the cable costs are minimized? In this paper, we show that this problem is NP-hard and present partitioning-based heuristics. Evaluation with different topologies demonstrate that our approach discovers better layouts than the previous approaches and reduces the cabling costs by up to 38%.**

I. Introduction

Designing a datacenter network that minimizes cost while satisfying performance requirements, on metrics such as bisection bandwidth and latency, is a hard problem with huge solution space [1]. A network designer has to consider a vast number of choices. For example, she has to choose from a number of topology families such FatTree [2,3], HyperX [4], BCube [5], DCell [6], and CamCube [7]. And each of these basic topologies has numerous parameters such as the number of interfaces per switch, size of the switch, and type of cables and connectors (e.g., optical or copper, 1G or 10G or 40G).

In our previous work [1], we proposed *Perseus*, a framework to assist network designers in systematically exploring the datacenter network design space to determine a cost-effective solution satisfying their requirements. Given bisection bandwidth and latency requirements, Perseus quickly quantifies the costs and significantly reduces the search space. That previous work exposed several novel optimization problems and proposed heuristics.

One important aspect of the datacenter network design that Perseus [1] exposed is the cabling problem: given a logical topology of switches, servers, and links, generate a feasible mapping of these elements onto a physical space with racks and cable trays such that the total wiring cost is minimized. During the evaluation study of Perseus, we found that the wiring alone can run into several millions of dollars even for an 8K server network. It can be as much as 34% of the datacenter network cost. Hence, minimizing the wiring costs can result in significant savings.

Wiring cost is reduced if switches and servers are placed in racks such that the length of cables needed to realize the links is minimized. In Table I, we present the list prices for different Ethernet cables that support 10G and 40G of bandwidths. Note that these are quantity-one list prices and

This work was conducted while the authors were at HP Labs, Palo Alto.

TABLE I
Cable prices (dollars) for various lengths, as compiled in [1]

	Single channel	Quad channel		
Length (M)	SFP+ copper	QSFP copper	QSFP+ copper	QSFP+ optical
1	45	55	95	—
2	52	74	—	—
3	66	87	150	390
5	74	116	—	400
10	101	—	—	418
12	117	—	—	—
15	—	—	—	448
20	—	—	—	465
30	—	—	—	508
50	—	—	—	618
100	—	—	—	883

could cost much less when purchased in bulk. However, in general, we observe that the price increases with length. Another significant observation is that the cheap copper cables have a short limited maximum distance span of about 10 meters because of the signal degradation. For larger distances, expensive optical cables have to be used. Employing shorter cables have several other advantages too: (a) they reduce the total weight of the cables and hence the load on the cable trays, (b) they are easy to replace and manage, and (c) they result in smaller cable bundles at any location in the datacenter and hence do not disrupt airflow as much as the large bundles, which has direct impact on the cooling costs.

Traditionally, network architects manually designed the layouts; but, this process is slow and can result in sub-optimal solutions. Also, this is feasible only when deciding layouts for one or few topologies but quickly becomes infeasible when poring through a huge number of topology choices.

Previous datacenter networking research did not address the wiring optimization problem for general topologies. Most of the previous work on topologies focused on finding logical topologies that achieve high bisection bandwidth or achieve a certain bandwidth with minimal number of switches [4,8]. Popa *et al.* [9] compared the costs of several data-center network architecture families, but they did not focus on optimizing the placement of switches in racks for different topologies. Farrington *et al.* in [10] considered the cabling issues for FatTree networks [3] where they replace a subset of switches and links with merchant silicon and hence some links become traces on circuit boards. However, they do not consider the problem of cabling arbitrary topologies.

The cabling problem has some similarities to the well-researched VLSI Cell Placement problem [11] where the goal is to place modules of a given electrical circuit and to lay out the wires connecting the modules in a two-dimensional space such that the wire length and the layout area is minimized. There are some differences between this problem and our cabling problem. For example, because the wires have to be etched on to a circuit board, the layout of wires has severe constraints in the VLSI Cell Placement problem. However, the techniques proposed for finding feasible solutions for the VLSI Cell Placement problem are applicable to our problem and we indeed leverage the partitioning based placement method in our approach.

We make the following contributions in this paper:

1) We formalize the datacenter network cabling problem. Such a formalism helps us in attacking the problem in a fairly general setting, without restricting ourselves to any specific topology.
2) We show that the cabling problem is NP-hard.
3) We propose a *hierarchical partitioning* heuristic, which maximizes the use of shorter and hence cheaper cables.
4) We evaluate the performance of our algorithm on several different topologies and compare the performance to the simple heuristic presented in [1]. We observe that our technique can reduce the cabling costs by up to 38%.

Roadmap. The rest of the paper is organized as follows. We give a brief description on our physical space and the logical topology, and describe the graphical representations for the two in §II. We also formally define the problem statement in §II. We settle the computational complexity of the most general form of our problem; and give a high-level overview of our technique in §III. In §IV, we give a detailed description of our technique to partition the physical topology. The algorithms for partitioning the logical topology are given in §V; we finish the algorithmic description by describing the placement and cabling technique in §VI. We present evaluation results for our algorithm in §VII and close the paper with some future directions in §VIII.

II. Problem

We will provide more details of the cabling problem in this section. As mentioned in the previous section, the goal of our work is to lay out a given logical topology on to a physical space such that the total wiring cost is minimized.

A. Physical space

Datacenters are typically organized as rows of racks. Standard-size racks have fixed widths and are divided on the vertical axis into Rack Units (RUs) of height 1.75 inches. Rack heights vary from 16RU to 50RU, but most common rack height sold for datacenters is 42RU [12]. For running cables between RUs in a rack or cables exiting the rack, each rack has plenum space on either side of the rack. Thus cables are run from the face plate to the side on the either end. This also ensures that the cables do not block the air flow and hence do not affect the cooling.

While racks in a row are placed next to each other, two consecutive rows are separated by either a "cold aisle" or an "hot aisle". A cold aisle is a source of cool air and an hot aisle is a sink for heated air. Several considerations govern the choice of aisle widths [13], but generally the cold aisle must be at least 4 feet wide and the hot aisle at least 3 feet wide.

In modern data centers, network cables do not run under raised floors, because it becomes too painful to trace underfloor cables when working on them. Therefore, inter-rack cables run in ceiling-hung trays which are few feet above the racks. One tray runs directly above each row of racks, but there are relatively few trays running between rows, because too many cross trays can restrict air flow.

Given a datacenter layout with rows of racks, to connect servers or switches (elements) at two different RUs, u_1 and u_2, one has to run a cable as follows. First, run the cable from the faceplate of the element at u_1 to the side of the rack. If both u_1 and u_2 are in the same rack, then the cable need not exit the rack and just need to be laid out to the RU u_2 and then to the faceplate of the element at u_2. If u_1 and u_2 are in two different racks, then the cable has to exit the rack of u_1 and run to the ceiling-hung cable tray. The cable then need to be laid on the cable tray to reach the destination rack. Since cross trays may not run on every rack, the distance between the top of two racks can be more than the simple Manhattan distance [1]. Once at the destination rack, the cable is run down from the cable tray and run on the side to the RU u_2.

We use function $d(u_1, u_2)$ to denote the cable length required to connect RUs u_1 and u_2.

Given a physical space with a number of rows and number of racks per rows, we model it as a complete undirected graph $\mathcal{H}$ with each RU of every rack in the physical space as a vertex. We refer to $\mathcal{H}$ as the host graph. Links in the host graph are weighted: the weight of a link between two vertices u_1 and u_2 is $d(u_1, u_2)$. To track the mapping between nodes in the host graph and RUs of the physical space, we define two functions $r(\cdot)$ and $u(\cdot)$, that maps each node in the host graph to the corresponding rack and RU in that rack, respectively.

B. Logical topology

Logical topology consists of several switches and servers and links between them. So, we model it as an undirected graph $\mathcal{G}$, which we refer to as the guest graph, with switches and servers as the vertices in the guest graph and the links between them as the edges. There can be multiple links between two elements (switches or servers) in the logical topology. Thus, to account for multiplicity of links, we associate a weight w with each edge that represents the number of links between the corresponding elements in the logical topology.

Datacenter switches and servers come in different form factors. Typically, switches span standard-size rack width but may be one or more RU in height. However, compute nodes such as blades come in variety of forms, but we can model them as having a fraction of RU. For example, a configuration where a two blades side-by-side occupy a RU, we can model them as each having a size of 0.5RU. To handle different form factors, for each node in the guest graph, we have a size s associated with it which corresponds to the height of the corresponding physical element.

C. Formulation

We start by defining the mapping problem formally. We will need the notion of a mapping function. Given an arbitrary $\mathcal{G}$, and an arbitrary $\mathcal{H}$, a **mapping function** f is a function that maps each node v in $\mathcal{G}$ to a subset of nodes in $\mathcal{H}$ such that the following hold true:

1) Size of v is less than or equal to the size of $f(v)$, i.e.,
$$\forall v \in \mathcal{G}, \quad s(v) \leq f(v)$$
2) If size of v is greater than 1, then $f(v)$ consist of only nodes that are consecutive in the same rack i.e.,
$$\forall v \in \mathcal{G}, \forall i,j \in f(v), r(i) = r(j) \text{ and }$$
$$|u(i) - u(j)| < |f(v)|$$
3) no node in the host graph is overloaded i.e.,
$$\forall h \in \mathcal{G}, \sum_{v \in V_h} s(v) \leq |\cup_{v \in V_h} f(v)|$$
$$\text{where } V_h = \{v \in \mathcal{G} | h \in f(v)\}$$

The cost of a mapping function, denoted by cost(f), is defined as the sum, over all edges in $\mathcal{G}$, the cost of the cables required to realize those edges in the physical space under the mapping function f. To accomodate nodes in $\mathcal{G}$ with size greater than one, we define function $f'(v) = \arg\min_{w \in f(v)} u(w)$ that computes the lowest height RU assigned to the node v. Thus, formally, the cost cost(f) is defined as follows:

$$\text{cost}(f) = \sum_{(v_1,v_2) \in \mathcal{G}} d(f'(v_1), f'(v_2)) + s(v_1) + s(v_2) - 2$$

We add the sizes of nodes v_1 and v_2 in the cost function as the cable may start and end anywhere on the faceplate of the respective physical elements.

We are now ready to formally state our problem.

Problem (L2P-MAP): Given an arbitrary $\mathcal{G}$, and an arbitrary $\mathcal{H}$, find a mapping function f with the minimal cost.

D. Computational Complexity

In this subsection, we prove that the mapping problem is NP-hard for the case when one is given an arbitrary logical topology and a linear physical space. We start by recalling the minimum linear arrangement problem [14]–[16]:

Problem (Minimum Linear Arrangement Problem (MLAP)): Given an undirected graph $G = (V, E)$, with $|V| = n$, the minimum linear arrangement problem is to find a one-to-one function $\psi : V \to [1, 2, \ldots, n]$, so as to minimize
$$\sum_{(u,v) \in E} |\psi(u) - \psi(v)|$$

The MLAP is known to be an NP-hard problem [17]. In order to prove the NP-hardness of the L2P-MAP problem defined in §II, we reduce the MLAP problem to L2P-MAP.

Given an instance of MLAP, we create an instance of L2P-MAP as follows. We start by constructing the guest graph $\mathcal{G}$ for the L2P-MAP problem. The vertex set and the edge set of $\mathcal{G}$ is same as that of the undirected graph G in the MLAP problem. Each node in $\mathcal{V}$ is given size 1 and each edge in $\mathcal{E}$ is given weight 1. In order to construct the host graph $\mathcal{H}$, we create $n = |V|$ nodes and number the nodes from 1 to n; for $1 \leq i < n$, each node i has an outgoing edge to a node $(i+1)$ that is of weight 1.

The L2P-MAP problem is then to find a mapping function that maps each node in $\mathcal{G}$ to a subset of nodes in $\mathcal{H}$ such that the three conditions stated in §II are satisfied. Given the reduction above, we make three observations: first, the resulting mapping function has to be a one-to-one function, otherwise it will violate condition (1) in §II; second, the cost function as stated in §II reduces to
$$\text{cost}(f) = \sum_{(v_1,v_2) \in \mathcal{E}} d(f(v_1), f(v_2));$$
and finally, given the construction of the host graph above, the distance function for the host graph is given by $d(u, v) = |u - v|$.

Using the above three observations, it is easy to see that the mapping function f for the above instance of L2P-MAP problem is indeed the function ψ for the MLAP problem. Hence, an algorithm that computes an optimal solution for the L2P-MAP problem can be used to compute an optimal solution for MLAP problem. Furthermore, since the above reduction is a polynomial-time reduction, an algorithm that solves L2P-MAP problem in polynomial time can be used to solve MLAP in polynomial time. Hence, NP-hardness of MLAP problem implies that L2P-MAP problem is NP-hard.

The above reduction has the following implications. Given an arbitrary logical topology, it is NP-hard to find a mapping function that minimizes the cabling cost even if the racks in the physical space are arranged as a planar graph, a tree or even a linear array.

III. Our technique

Having proved that it is computationally hard to solve the general L2P-MAP problem, we turn our attention to designing algorithms that may work well in practice. We start by giving an overview of our approach to tackle this problem.

A. Overview of our algorithm

Before describing our technique, we state an assumption that we make throughout the paper. We assume that we are given

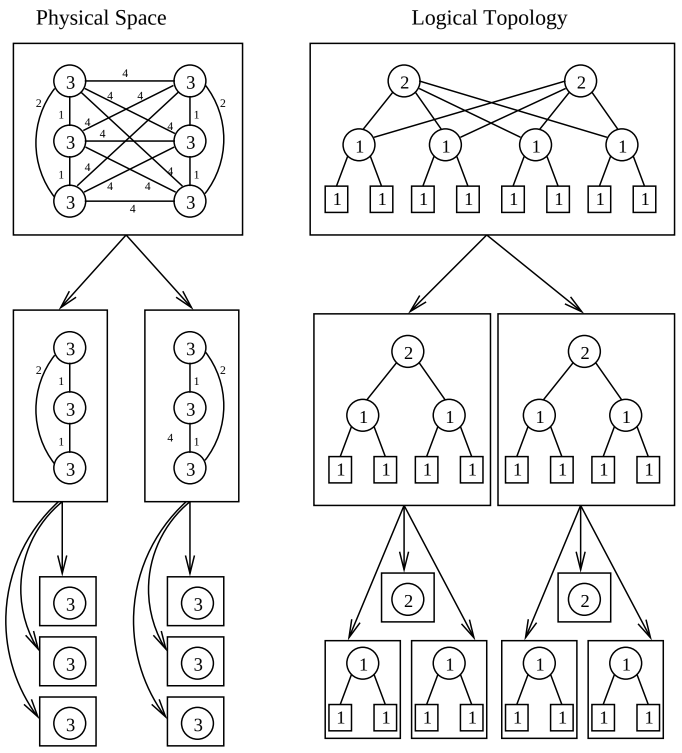

Fig. 1. Example for hierarchical partitioning of a physical space and matching hierarchical partitioning of a given logical topology. For the physical space, the number inside a node denotes the capacity of the node and the number on a link denotes the distance between corresponding nodes. For the logical topology, we assume all links are of unit weight and the number inside a node denotes the size of the node.

a set of k available cable types with different fixed lengths $\ell_1, \ell_2, \ell_3, ..., \ell_k$ where $\ell_i < \ell_j$ for $1 \leq i < j \leq k$. We suppose that ℓ_k can span any two furthest RUs in the datacenter. We assume that a longer cable costs more than a shorter cable (See Table I).

Our objective is to design a mapping function that has a low cost. A key observation to minimize cost is that (sets of) nodes in the logical topology that have dense connections (larger number of edges between themselves) should be placed physically close in the physical space, so that the lower cost cables can be used. We leverage this and employ a partitioning scheme to map the nodes in the logical topology to the physical space.

In order to reduce the problem size, we consider a compacted host graph where each node corresponds to a rack instead of a RU (more details in §IV). We hierarchically partition the host graph into k levels such that the nodes within the same partition at a level i can be wired with the cables of length ℓ_i. We perform partitioning at each level generating maximal-sized partitions. We show a toy example of physical space hierarchical partitioning on the left hand side in Figure 1. We consider a datacenter with 6 racks, each with 3 RU height, spread apart with distances as shown in the figure. We assume that we have three cable types available with lengths 0.5, 3, and 10.

Algorithm 1 Overview of our technique
```
1: Inputs: Logical Topology, Physical space
2: Hierarchically partition the physical space
3: Generate a matching partition for the logical topology
4: Starting with the coarsest partitioning level
5:     Map the logical partitions to the physical partitions
6: Place nodes in the leaf partitions on to the racks
7: Identify cabling
```

Second, we generate a *matching* hierarchical partition of the logical topology into k levels such that each partition of the logical topology at a level i can be placed in a level i partition of the physical space. While partitioning the logical topology at different levels, we maximize the number of links that are included in the partitions (intra-partition links) and this ensures that the number of shorter cables used is maximized. Top right hand side of the Figure 1 shows a logical topology—a two-level fattree with 8 servers. Here we assume all links are of unit weight. Core switches are 2RU in size and all other switches and servers are 1U in size (number shown inside the node). We also show a minimal hierarchical partitioning of this logical topology that matches with the hierarchical partitioning of the physical space on the left side.

Once we have the partitioning of the physical space (that exploits proximity) and of the logical topology (that exploits connectivity), the final task is the actual placement of nodes in the logical topology into the physical space defined by the cluster that they are mapped to. while we exploit proximity of RUs and connectivity of logical nodes The overall algorithm is shown in Algorithm 1.

IV. Partitioning the physical space

In this section, we describe our technique for partitioning the physical space.

Recall from §III that our objective for partitioning the physical space is to exploit the locality of rack units (RU). That is, we want to identify a set of clusters such that any two RUs within the same cluster can be connected using cables of a specified length, and RUs in different clusters may require longer cables.

We can significantly simplify the partitioning problem by observing that RUs within the same rack can be connected using short cables – any two RUs in a rack require cables of length at most 3 meters. Therefore, we can always place all RUs within one rack into the same partition.

In order to exploit this observation, we modify the physical space modeling described in §II as follows. Instead of a node per RU in the host graph $\mathcal{H}$, we model each rack as a node in $\mathcal{H}$. We associate a capacity with each node that denotes the number of RUs in each rack. The weight of a link between two nodes is set as the length of the cable required to wire between the bottom RUs of the corresponding racks.

In order to succinctly describe the partitioning algorithm, we use the notion of r**-decompositions**. For a parameter r, an r-decomposition of a weighted graph H is a partition of the vertices of H into clusters, with each cluster having a diameter at most r. There are a number of algorithms that can be used to generate r-decompositions of a given graph [18,19]; our technique is amenable to any of these algorithms. For evaluation purpose, we use a simple algorithm to construct an r-decomposition of a given weighted graph H, that is shown in Algorithm 2.

Algorithm 2 Constructing an r-decomposition

```
1: Inputs: graph $H$, integer $r$
2: Unmark all nodes in $V(H)$
3: While not all nodes marked
4:     Select an unmarked node $u \in V(H)$ with least ID
5:     Let $C = \{v \in V(H) \mid v$ unmarked; $d(u,v) \leq r\}$
6:     Mark all nodes in $C$ and make it a new cluster
```

We also note that our technique is essentially oblivious to the actual structure of the physical space – separation between racks, aisle widths, how cross trays run across the racks and the rows, etc. As long as there is a meaningful way to define a distance function (edge weights in the host graph) and the corresponding distances adhere to the requirement of the underlying algorithm to generate an r-decomposition, the above algorithm will generate r-decompositions.

Algorithm 3 Partitioning the physical space

```
1: Inputs: Physical space $H$
2: length of cables $\{\ell_1, \ldots, \ell_k\}$
3: $P_{k+1} \leftarrow V(H)$
4: For $i = k$ downto 1
5:     For each cluster $C$ in $P_{i+1}$
6:         Construct an $\ell_i$-decomposition of $C$
7:         Let the set of new clusters be $P_i$
```

We now discuss the hierarchical partitioning technique for the physical space. The algorithm, shown in Algorithm 3, takes in two inputs – the physical space and the set of available cables. It partitions the physical topology in a top-down fashion – that is, it starts by initializing the complete set of nodes to be a single highest level cluster. It then, recursively for each given cable length in decreasing order, partitions each cluster at the higher level into smaller clusters that have a diameter at most the length of the cable used to partition at that level. More formally, for generating clusters of diameter ℓ_t, it computes the ℓ_t-decomposition for each cluster at level $t+1$.

In order to make the following discussion succinct, we assume throughout the paper that $\ell_1 = 0$. That is, the lowest level partitions correspond to a single rack in the physical space. Although our technique does not exploit this assumption, discussion of our algorithms on how to actually place the logical nodes in the physical space becomes simplified.

V. Partitioning the logical topology

In this section, we describe our technique for partitioning the logical topology. In contrast to the partitioning technique of the physical space that exploited the proximity of the rack units, our technique for partitioning the logical topology generates partitions such that nodes within a single partition are expected to be densely connected. We formalize the notion of dense connections below; followed by a detailed discussion of our techniques and close the section with a brief discussion on the implementation challenges.

As mentioned in §II, we model the logical topology as an arbitrary weighted undirected graph $\mathcal{G}$, with each edge having a weight representing the number of links between the corresponding logical nodes. We make no assumptions on the structure of the logical topology; this allows us to design placement algorithms for a fairly general settings – irrespective of whether the logical topology has a structure (fat tree topology, for instance) or is completely unstructured (random graph topologies [20], or [21]). We do, however, acknowledge that it may be possible to exploit the structure of the logical topology for improved placement.

We start by formally stating the problem statement that we intend to present algorithms for.

A. Problem statement

Given a hierarchical partitioning P_p of the physical space, our goal is to generate a *matching* hierarchical partitioning of the logical topology P_l, while minimizing the cumulative weight of the inter-partition edges at each level. We define an hierarchical partition P_l matches another hierarchical partition P_p if they have same number of levels and there exists an injective mapping of each partition p_1 at each level ℓ in P_l to a partition p_2 at level ℓ in P_2 such that the size of p_2 is greater than or equal to the size of p_1.

We generate matching partitions in a top-down recursive fashion. At each level, we solve several *partitioning* sub-problems. At the top most level, we need to solve only one partitioning sub-problem: to partition the whole logical topology into partitions that matches the partitions of the physical topology at the top level. At other levels, we need to run as many partitioning sub-problems as there are the number of logical node partitions.

We define the partitioning sub-problem as follows. Suppose $p_1, p_2, \ldots, p_k$ be the sizes of k partitions that we target to match during a partitioning sub-problem. Given a connected, weighted undirected graph $L = (V(L), E(L))$, partition $V(L)$ into clusters $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \emptyset$ for $i \neq j$, $|V_i| \leq p_i$, and $\cup V_i = V(L)$ such that the weight of edges in the *edge-cut* – defined as the set of edges that have end points in different partitions – is minimized.

Although the partitioning problem is known to be NP-hard [22], there are a number of algorithms that have been designed due to its applications in the VLSI design and, multiprocessor scheduling, and load balancing. The main technique used in these algorithms is multilevel recursive partitioning [22]. We leverage one of the available graph partitioning tools [23] that generates efficient partitions by exploiting multilevel recursive partitioning along with several heuristics to improve the initial set of partitions. However, our constraints for the problem are far more stringent than the ones used in this tool – we describe these implementation issues after giving a high level description of the subroutines from the tool that we used for our purposes.

B. Generating the partitions

The METIS software provides a set of subroutines for solving the partitioning sub-problem. We briefly describe the specific functionalities provided by the software that we used in our implementation; for a detailed discussion, please refer to [23].

The partitioning algorithm works in three steps. In the first step, the size of the graph is reduced in such a way that the edge-cut in the smaller graph approximates the edge-cut in the original graph; this is achieved by collapsing the vertices that are expected to be in the same partition into a *multi-vertex*. The weight of the multi-vertex is the sum of the weights of the vertices that constitute the multi-vertex. The weight of the edges incident to a multi-vertex is the sum of the weights of the edges incident on the vertices in the multi-vertex. Using such a technique allows us to reduce the size of the problem without distorting the edge-cut size; that is the edge-cut size for partitions of the smaller instance is *exactly* equal to the edge-cut size of the corresponding partitions in the original problem. In order to collapse the vertices, we used the *heavy-weight matching* heuristic. In this heuristic, a maximal matching of maximum weight is computed using a randomized algorithm and the vertices that are the end points of the edges in the matching are collapsed; earlier analysis shows that this heuristic, although suboptimal, works well in practice [22].

The new graph generated by the first step is then partitioned using a brute-force technique, which we describe next. Note that since the size of the new graph is sufficiently small, a brute-force approach leads to efficient partitions within reasonable amount of processing time. In order to match the partition sizes, a greedy algorithm is used to partition the smaller graph. In particular, the algorithm start with an arbitrarily chosen vertex and *grows a region* around the vertex in a breadth-first fashion, until the size of the region corresponds to the desired size of the partition. Since the quality of the edge-cut of so obtained partitions is sensitive to the selection of the initial vertex, several iterations of the algorithm are run and the solution that has the minimum edge-cut size is selected.

In the final step, the partitions thus generated are projected back to the original graph. During the projection phase, another optimization technique is used to improve the quality of partitioning. In particular, the partitions are further refined using the Kernighan-Lin algorithm [24], with the modifications proposed by Fiduccia and Mattheyses [25] – a heuristic often used for graph partitioning with the objective of minimizing the edge-cut size. Starting with an initial partition, the algorithm in each step searches for a subset of vertices, from each part of the graph such that swapping these vertices leads to a partition with smaller edge-cut size. The algorithm terminates when no such subset of vertices can be found or a specified number of swaps have been performed.

C. Implementation issues

There are two implementation issues with using the tool that we outline next. First, the subroutine requires that the number of nodes in the input graph be equal to the sum of sizes of the partitions specified in the input. This caused a potential inconsistency in our implementation because the desired size of the partitions (generated by partitioning the physical space) are a factor of the size of the physical racks, which may have little correspondence to the number of servers and switches required in the logical topology. In order to overcome this problem, we added extra nodes in the logical topology; these nodes were forced to have no outgoing edges and were given weight 1. After the completion of the placement algorithm, these extra nodes correspond to unused rack units in the racks that they are assigned to.

Second, the subroutines provided by the tool use heuristics that may generate partitions of sizes that are only an approximation and not exact to the partition sizes specified in the input to the subroutine. This may lead to consistency problems when mapping the logical topology on to the physical space. In order to overcome this issue, we designed a simple Kernighan-Lin style algorithm to *balance* the partitions. Our algorithm computes, for each node in a partition A that has a larger size than desired, the cost of moving the node to a partition B that has a smaller size than desired. The cost is defined as the increase in the number of inter-cluster edges if the node were moved from A to B. It then moves the node with the minimum cost from A to B. Since all nodes have unit weights during the partitioning phase, the algorithm is always able to balance the distorted outputs generated by the subroutine to the desired ones.

VI. Placement and Cabling

Once we have the partitions for the physical space, and the corresponding matching partitions of the logical topology, we have two remaining tasks before we can decide the exact locations for each switch and server in the logical topology. First, we need a placement algorithm for placing a set of logical nodes assigned to a rack into the rack units of the rack (§VI-A). Second, we need to identify the exact cables needed to realize all links in the logical topology and compute the cabling cost (§VI-B).

A. Placement on rack units

Recall that our partitioning algorithm for the physical space uses the modified host graph with racks as nodes. This results in the lowest level clusters being at the granularity of the racks. Hence, the logical topology partitioning described in the previous section essentially assigns each node in the logical topology to a rack; the function is many-to-one, that is, several logical nodes may be assigned to the same rack by the mapping function. Our next task is to place these logical nodes on to the RUs on the rack they are assigned to.

We describe a simple algorithm for the placement of logical nodes on to the rack units. Recall, from §II, that our description of the physical space details that the inter-rack cables run at the top of the rack. Hence, in order to reduce the cable length, we would like to place the logical nodes that have more links to logical nodes in the other clusters on the top of the rack. This observation is at the heart of our placement algorithm (see Algorithm 4).

Algorithm 4 Placement of nodes

```
1: Inputs: Logical topology $G$; nodes $V_R$ on a rack $R$
2: For each node $v \in V_R$ on rack $R$
3:     Compute the weight of links to nodes in $V(G) \setminus V_R$
4: Sort the nodes in decreasing order of outgoing links
5: Starting with the first node
6:     Place the node at the topmost available position
```

The algorithm takes as input the graph representation of the logical topology G, a rack R and a set of logical nodes V_R that are assigned to rack R by the mapping function. The algorithm starts by computing, for each node in V_R, the weight of the logical links to the nodes that are assigned to a rack other than R. For any node $v \in V_R$, given the logical topology and the set V_R, this can be easily computed by iterating over the set of edges incident on v, and checking if the other end of the edge is in V_R or not.

Once the weight of links to nodes on other racks is computed for each node, we sort the nodes in decreasing order of these weights. The idea is to greedily place the node, among the remaining nodes, with maximum weight of links to other racks at the top most available position on the rack.

B. Cabling

Once matching partitions are generated and placement is decided, determining the cable to use for realizing each link in the logical topology is straightforward. After partitioning and placement, a unique RU in the physical space is assigned for each node in the logical topology. We compute the minimum length of the cable needed to realize each link of the logical topology, using the function $d(\cdot, \cdot)$, as described in §II. Then we pick the shortest cable type from the set of cable types $\ell_1, \ell_2, ..., \ell_k$ that is greater than the minimum cable required. We use the price for this cable in computing the total cabling cost.

One aspect to note is that the cabling is decided based on the final placement of the nodes and not based on how partitioning is done. Observe that two logical nodes that have a link between them and are in different partitions at level i may indeed be finally wired with a cable of length $\ell_j < \ell_i$.

VII. Evaluation

In this section, we present evaluation results for our hierarchical placement technique. We start by discussing the evaluation methodology (§VII-A), followed by a detailed discussion on the evaluation results (§VII-B).

A. Methodology

In this subsection, we give specific details on the evaluation set up.

Physical space. We start by describing the details of the physical space used in our simulations. The racks are assumed to be 24 inches wide and 78 inches tall. Each RU is 1.75 inches, leading to 42 RUs packed within a rack. The racks are arranged in rows, with aisle width being 48 inches (4 feet). The ceiling-hung trays and the cross-trays are 24 inches above the racks; with cross-trays running along every second rack along the rows. Using these dimensions, we can generate the mesh representation of the physical topology by using the distance function $d(\cdot, \cdot)$ discussed in [1].

Logical topologies. We present evaluation results for two logical topologies: (a) A fat-tree topology [3]; and, (b) A HyperX topology [4]. There are two main factors that may potentially effect the placement of logical nodes on to the physical space. First, the server capacity of the data-center – for a fixed physical space, increasing the size of the data-center (while keeping fixed bisection bandwidth) will potentially effect the placement problem. Table II presents the details on the Fattree topology used in our evaluation results (along with the physical space dimensions for each topology); all these topologies have full bisection bandwidth and were assumed to have switches with radix 48.

TABLE II
Logical topology and physical space dimensions used in our evaluation

# servers	# switches	# racks	Arrangement of racks
1152	120	30	5 × 6
2304	240	65	5 × 13
4608	480	130	10 × 13
9216	960	250	10 × 25
18432	1920	500	20 × 25
27648	2880	750	25 × 30

Second, even for a logical topology with fixed number of servers, varying the oversubscription ratio may potentially lead to different placement strategies. Intuitively, this is due to the fact that placement schemes are expected to be sensitive to the connectivity of logical nodes. In order to evaluate this, we fix the number of servers in the logical topology to $27,648$ and change the oversubscription ratio from $1:1$ to $4:1$; we keep the number of racks used fixed to 750 (as in Table II) since varying the oversubscription ratio does not change the total number of logical nodes significantly.

Cables. For our evaluation purposes, we use cables of varying length as described in Table I. For each cable of a particular length, we choose the minimum cost among all available cable types for that length.

B. Results and Discussions

We compare the performance of our technique to a greedy placement technique. Such a greedy placement was proposed in [1,3]. We start by briefly describing the greedy placement heuristic.

TABLE III
CONFIGURATIONS FOR THE HYPERX TOPOLOGIES USED IN OUR EVALUATION; NUMBER OF SERVERS ARE ALWAYS 8192; NUMBER OF RACKS USED IS 210 FOR EACH CONFIGURATION WITH A 10×21 ARRANGEMENT.

Oversub ratio	#terminals per switch	#dim	# switches in each dimension	Lag factor in each dimension	Cost of Greedy scheme	Cost of Hierarchical technique	Cost Ratio of Hierarchical to greedy
32:1	32	6	(2, 2, 2, 2, 2, 8)	(1, 1, 1, 1, 1, 1)	3502293	3094325	0.88
32:1	32	4	(2, 2, 8, 8)	(1, 1, 1, 1)	3690717	3190140	0.86
16:1	32	8	(2, 2, 2, 2, 2, 2, 2, 2)	(2, 2, 2, 2, 2, 2, 2, 2)	3270906	3332908	1.02
16:1	32	4	(4, 4, 4, 4)	(1, 1, 1, 1)	3474500	3041148	0.88
8:1	16	3	(2, 16, 16)	(2, 1, 1)	6040032	5409483	0.90
16:3	16	9	(2, 2, 2, 2, 2, 2, 2, 2, 2)	(3, 3, 3, 3, 3, 3, 3, 3, 3)	3679746	3282017	0.89
16:3	16	5	(2, 2, 2, 4, 16)	(3, 3, 3, 2, 1)	5116673	5112202	1.00
4:1	16	7	(2, 2, 2, 2, 2, 2, 8)	(4, 4, 4, 4, 4, 4, 1)	4241860	3693705	0.87
4:1	16	3	(8, 8, 8)	(1, 1, 1)	4960092	4826893	0.97

The greedy placement heuristic is based on the observation that packing servers along with their edge switches may lead to a good placement scheme. It packs the set of servers attached to a particular edge switch, along with the switch itself, into one rack, as long as the size of the rack permits so; if the remaining space on a rack does not allow packing an edge switch along with all its servers, it greedily packs as many servers as possible on to the rack and places the remaining servers and the edge switch on a new rack. Once all the servers and edge switches are packed, aggregation and core switches are packed as tightly as possible on to the next available rack.

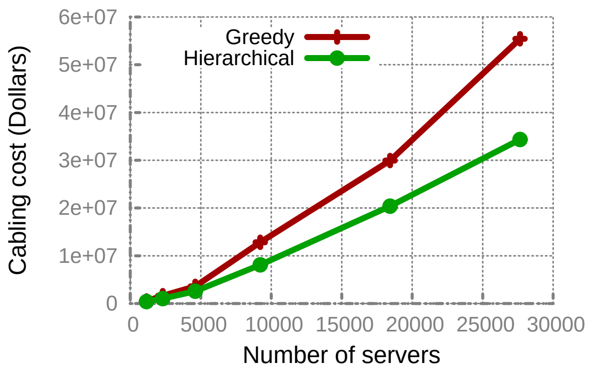

Fig. 2. Comparison of cabling costs for logical topologies with $1 : 1$ oversubscription ratio and varying server capacity.

Figure 2 compares the performance of our hierarchical placement technique to the the greedy scheme. We make two observations. First, our hierarchical placement technique consistently outperforms the greedy placement scheme with increasingly higher cost savings as the size of the logical topology increases. As an interesting data point, for the fattree topology designed in [3] (27648 servers and $1 : 1$ oversubscription ratio), the hierarchical placement technique results in almost 38% reduction in cabling costs.

Next, we compare the performance of the greedy and hierarchical placement techniques for logical topologies with varying oversubscription ratio (see Figure 3). We observe that for lower oversubscription ratios, hierarchical placement technique has much better performance when compared to the greedy placement. For instance, we observe a 38% reduction in cabling costs for logical topologies with $1 : 1$ oversubscription ratio when compared to 20% reduction for $4 : 1$ oversubscription ratio. Lower oversubscription ratios are, in fact, of key interest since they are more likely to be the design point for data-center topologies.

In hindsight, such a variation in performance as oversubscription ratio is varied is what one would expect. The key point to notice is that logical topologies with lower oversubscription ratio have much higher connectivity (larger number of logical connections) between the edge, aggregation and core switches. Hence, the greedy scheme that packs edge switches with their servers may lead to inefficient placement, while the hierarchical scheme that adapts to the connectivity of the logical nodes is more cost effective.

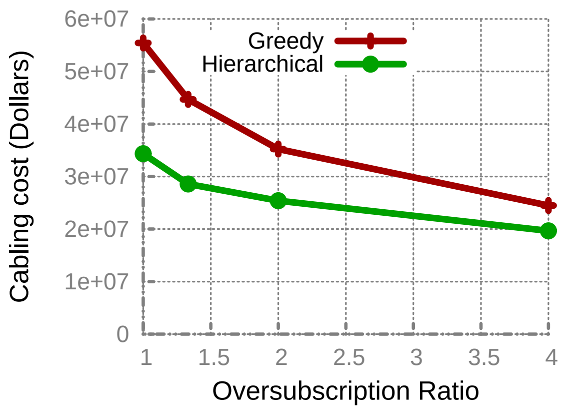

Fig. 3. Comparison of cabling costs for logical topologies with fixed server capacity and varying oversubscription ratio.

From the evaluation with FatTree topologies, the main take away is that our hierarchical placement technique naturally adapts to the structure of the logical datacenter topologies, and may be used as an effective initial placement strategy before applying topology specific optimizations.

For the HyperX topology, we generated multiple configurations by varying the oversubscription ratio, number of terminals per switch, dimension of the topology and the number of switches and links along various dimensions. HyperX topologies are not as easily extendible as FatTree topologies; hence, we could not do similar evaluation as in FatTree topologies. For FatTree topologies, it is easy to scale up or scale down the servers supported or the bisection bandwidth with just varying a single parameter. Instead, we compare the performance of greedy scheme and our hierarchical partitioning scheme with several configurations explored by our Perseus tool [1]. The results are shown in Table III for all the configurations.

We make several interesting observations that are in sharp contrast to the results for the FatTree topology. First, the reduction in cost for the HyperX topologies using the hierarchical partitioning scheme is significantly lesser when compared to that in FatTree topologies. For instance, the maximum reduction in cabling cost using the hierarchical partitioning scheme is close to 14%. Second, the reduction in cost for HyperX topologies is also independent of the oversubscription ratio of the datacenter network. It is likely that for these topologies, independent of the dimension and of the bisection bandwidth, it makes sense to always pack the servers (within the same rack) along with the edge switches - precisely what the greedy algorithm does.

VIII. Conclusion

Given a desired number of servers and required network performance, a network design engineer faces a humongous number of choices to make such as picking appropriate topology, types of switches to use, and type of cables to use, and has to layout the network, such that the overall cost is minimized. Our previous work, Perseus, assists networks designers in exploring through the design space in a systematic fashion. One of the open optimization problems from our previous work is to compute layout of servers, switches, and links in a physical space with racks such that overall cable cost is minimized. In this work, we formally define this cabling problem, study its computational complexity, and propose a hierarchical partitioning algorithm to determine a low-cost solution. Through evaluations with different topologies, we demonstrate that our algorithm lowers the cabling cost by up to 38% in comparison to a previously proposed greedy approach.

We outline some of the problems that this paper did not touch upon. First, our current technique does not take into account the plenum capacities of the racks. Our current work is focused on extending our technique to handle plenum capacities. Next, it would be interesting to evaluate and compare the performance of the hierarchical placement technique for other cost models; one such example is the labor centric cost model of Popa et. al. [9].

References

[1] J. Mudigonda, P. Yalagandula, and J. C. Mogul, “Taming the flying cable monster: A topology design and optimization framework for datacenter networks,” in *Proc. USENIX Annual Technical Conference (ATC)*, Portland, USA, June 2011.

[2] C. Clos, “A Study of Non-Blocking Switching Networks,” *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *Proc. ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, 2008.

[4] J. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, “HyperX: Topology, Routing, and Packaging of Efficient Large-Scale Networks,” in *Proc. Supercomputing*, Nov. 2009.

[5] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, “BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers,” in *Proc. ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, Barcelona, 2009.

[6] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, and S. Luz, “DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers,” in *Proc. ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, Aug. 2008.

[7] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O’Shea, and A. Donnelly, “Symbiotic Routing in Future Data Centers,” in *Proc. ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM)*, 2010.

[8] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufman, 2004.

[9] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, “A Cost Comparison of Data Center Network Architectures,” in *Proc. CoNEXT*, Dec. 2010.

[10] N. Farrington, E. Rubow, and A. Vahdat, “Data Center Switch Architecture in the Age of Merchant Silicon,” in *Proc. Hot Interconnects*, 2009, pp. 93–102.

[11] K. Shahookar and P. Mazumder, “VLSI Cell Placement Techniques,” *ACM Computing Surveys*, 1991.

[12] http://www.server-rack-online.com/select-rack-by-height.html.

[13] N. Rasmussen and W. Torell, “Data Center Projects: Establishing a Floor Plan,” American Power Conversion, White Paper 144, 2007.

[14] D. Adolphson and T. C. Hu, “Optimal linear ordering,” *SIAM Journal on Applied Mathematics*, vol. 25, no. 3, pp. 403–423, November 1973.

[15] J. Diaz, “Graph layout problems,” in *Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Prague, Czechoslovakia, August 1992, pp. 14–23.

[16] J. Díaz, J. Petit, and M. J. Serna, “A survey of graph layout problems,” *ACM Comput. Surv.*, vol. 34, no. 3, pp. 313–356, 2002.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[18] Y. Bartal, “Probabilistic approximations of metric spaces and its algorithmic applications,” in *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 1996, pp. 184–193.

[19] J. Fakcharoenphol, S. Rao, and K. Talwar, “A tight bound on approximating arbitrary metrics by tree metrics,” in *ACM Symposium on Theory of Computing (STOC)*, 2003, pp. 448–455.

[20] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, “Jellyfish: Networking data centers randomly,” in *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, June 2011.

[21] L. Gyarmati and T. A. Trinh, “Scafida: A scale-free network inspired by datacenter architecture,” *SIGCOMM CCR*, vol. 40, no. 5, pp. 5–12, 2010.

[22] 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.

[23] ——. METIS - serial graph partitioning and fill-reducing matrix ordering. http://glaros.dtc.umn.edu/gkhome/views/metis.

[24] B. W. Kernighan and S. Lin, “An Efficient Heuristic Procedure for Partitioning Graphs,” *The Bell system technical journal*, vol. 49, no. 1, pp. 291–307, 1970.

[25] C. M. Fiduccia and R. M. Mattheyses, “A linear-time heuristic for improving network partitions,” in *Proceedings of the 19th Design Automation Conference*, 1982, pp. 175–181.