How Hard Can It Be?

Understanding the Complexity of Replica Aware Virtual Cluster Embeddings

Carlo Fuerst[1,*] Maciej Pacut[2,*] Paolo Costa[3] Stefan Schmid[4]
[1] TU Berlin, Germany [2] University of Wroclaw, Poland [3] Microsoft Research, UK [4] TU Berlin & T-Labs, Germany
[*] authors contributed equally to this work

***Abstract*—Virtualized datacenters offer great flexibilities in terms of resource allocation. In particular, by decoupling applications from the constraints of the underlying infrastructure, virtualization supports an optimized mapping of virtual machines as well as their interconnecting network to their physical counterparts: essentially a graph embedding problem.**

However, existing embedding algorithms such as Oktopus and Proteus often ignore a crucial dimension of the embedding problem, namely *data locality*: the input to a cloud application such as MapReduce is typically stored in a distributed, and sometimes redundant, file system. Since moving data is costly, an embedding algorithm should be data locality aware, and allocate computational resources close to the data; in case of redundant storage, the algorithm should also optimize the *replica selection*.

This paper initiates the algorithmic study of data locality aware virtual cluster embeddings on datacenter topologies. We show that despite the multiple degrees of freedom in terms of embedding, replica selection and assignment, many problems can be solved efficiently. We also highlight the limitations of such optimizations, by presenting several NP-hardness proofs; interestingly, our hardness results also hold in uncapacitated networks of small diameter.

I. Introduction

Distributed cloud applications, such as batch-processing applications or scale-out databases, generate a significant amount of network traffic [25]. For instance, MapReduce consists of a network intensive shuffle phase, where data is transferred from the mappers to the reducers. In order to ensure a predictable application performance, especially in shared cloud environments, it is important to provide isolation and bandwidth guarantees between the virtual machines [37], e.g., by making explicit network reservations [5]. Accordingly, modern batch-processing applications provide the abstraction of entire *virtual networks* [25], defining both the virtual machines as well as their interconnecting network. The most prominent virtual network abstraction is the *virtual cluster* [5], [34].

Virtualized datacenters offer great flexibilities on where these virtual networks can be instantiated or *embedded*. In order to maximize the resource utilization in the datacenter, it is in principle desirable to map the virtual machines of a given virtual network as close as possible in the underlying physical network, as this minimizes communication costs (respectively, bandwidth reservations) [5], [34].

However, existing systems often ignore a crucial dimension of the virtual network embedding problem: the fact that the input data for a cloud application, consisting of atomic *chunks*, is typically distributed across different servers and stored in a distributed file system [6], [17], [32]. In order to properly minimize communication costs, an embedding algorithm should hence also be *data locality aware* [4], [23], [36], and allocate (or *embed*) computational resources close to the to be processed data. Moreover, in case of redundant storage (batch processing applications often provide a 3-fold redundancy [32]), an algorithm should also be aware of, and exploit, *replica selection* flexibilities.

Our Contributions. This paper initiates the formal study of data-locality and replica aware virtual network embedding problems in datacenters. In particular, we decompose the general optimization problem into its fundamental aspects, such as assignment of chunks, replica selection, and flexible virtual machine placement, and answer questions such as:

1) Which chunks to assign to which virtual machine?
2) How to exploit redundancy and select good replicas?
3) How to efficiently embed virtual machines and their inter-connecting network?
4) Can the chunk assignment, replica selection and virtual machine embedding problems be jointly optimized, in polynomial time?

We draw a complete picture of the problem space: We show that even problem variants exhibiting multiple degrees of freedom in terms of replica selection and embedding, can be solved optimally in polynomial time, and we present several efficient algorithms accordingly. However, we also prove limitations in terms of computational tractability, by providing reductions from 3-D matching and Boolean satisfiability (SAT). Interestingly, while it is well-known that (unsplittable) multi-commodity flow problems are NP-hard in capacitated networks, our hardness results also hold in *uncapacitated* networks; moreover, we show that NP-hard problems already arise in small-diameter networks (as they are widely used today [2]).

Organization. Section II introduces our formal model in detail. Algorithms are presented in Section III and hardness results are presented in Section IV. After discussing related work in Section V, we conclude our work in Section VI.

II. Model

To get started, and before introducing our formal model and its constituting parts in detail, we will discuss the practical motivation. Figure 1 gives an overview of our model.

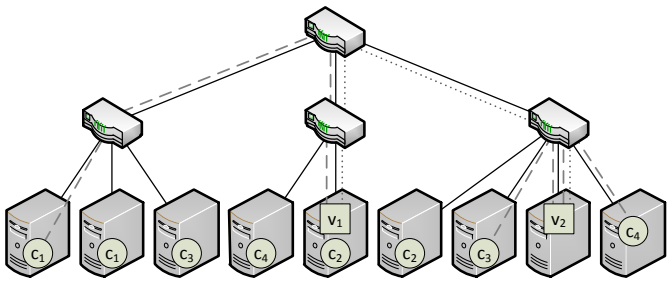

Fig. 1. Overview: a 9-server datacenter storing $\tau = 4$ different chunk types $\{c_1, \ldots, c_4\}$ (depicted as *circles*). The chunk replicas need to be selected and assigned to the two virtual machines v_1 and v_2; the virtual machines are depicted as *squares*, and the network connecting them to chunks (at bandwidth b_1) is *dashed*. In addition, the virtual machines are inter-connected among each other at bandwidth b_2 (*dotted*). The objective of the embedding algorithm is to minimize the overall bandwidth allocation (sum of *dashed* and *dotted* lines).

A. Background and Practical Motivation

Our model is motivated by batch-processing applications such as MapReduce. Such applications use multiple virtual machines to process data, often redundantly stored in a distributed file system implemented by multiple servers. [4], [11] The standard datacenter topologies today are (multi-rooted) fat-tree resp. *Clos* topologies [2], [20], hierarchical networks recursively made of sub-trees at each level; servers are located at the tree leaves. Given the amount of multiplexing over the mesh of links and the availability of multi-path routing protocol, e.g. ECMP, the redundant links can be considered as a single aggregate link for bandwidth reservations [5], [34].

During execution, batch-processing applications typically cycle through different phases, most prominently, a map phase and a reduce phase; between the two phases, a shuffling operation is performed, a phase where the results from the mappers are communicated to the reducers. Since the shuffling phase can constitute a non-negligible part of the overall runtime [8], and since concurrent network transmissions can introduce interference and performance unpredictability [37], it is important to provide explicit minimal bandwidth guarantees [25]. In particular, we model the virtual network connecting the virtual machines as a virtual cluster [5], [25], [34]; however, we extend this model with a notion of data-locality. In particular, we distinguish between the bandwidth needed between the assigned chunk and virtual machine (b_1) and the bandwidth needed between two virtual machines (b_2).

B. Formal Model

Let us now introduce our model more formally. It consists of three fundamental parts: (1) the substrate network (the servers and the connecting physical network), (2) the input which needs to be processed (divided into data chunks), and (3) the virtual network (the virtual machines and the logical network connecting the machines to each other as well as to the chunks).

The Substrate Network. The substrate network (also known as the *host graph*) represents the physical resources: a set S of $n_S = |S|$ servers interconnected by a network consisting of a set R of routers (or switches) and a set E of (symmetric) links; we will often refer to the elements in $S \cup R$ as the *vertices*. We will assume that the inter-connecting network forms an (arbitrary, not necessarily balanced or regular) tree, where the servers are located at the tree leaves. Each server $s \in S$ can host a certain number of virtual machines (available server capacity $cap(s)$), and each link $e \in E$ has a certain bandwidth capacity $cap(e)$.

The Input Data. The to be processed data constitutes the input to the batch-processing application. The data is stored in a distributed manner; this spatial distribution is given and not subject to optimization. The input data consists of τ different *chunk types* $\{c_1, \ldots, c_\tau\}$, where each chunk type c_i can have $r_i \geq 1$ instances (or replicas) $\{c_i^{(1)}, \ldots, c_i^{(r_i)}\}$, stored at different servers. A single server may host multiple chunks. It is sufficient to process one replica, and we will sometimes refer to this replica as the *active* (or selected) replica.

The Virtual Network. The virtual network consists of a set V of $n_V = |V|$ virtual machines, henceforth often simply called *nodes*. Each node $v \in V$ can be placed (or, synonymously, *embedded*) on a server; this placement can be subject to optimization.

Depending on the available capacity $cap(s)$ of server s, multiple nodes may be hosted on s. We will denote the server s hosting node v by $\pi(v) = s$. Since these nodes process the input data, they need to be assigned and connected to the chunks. Concretely, for each chunk type c_i, exactly one replica $c_i^{(j)}$ must be processed by exactly one node v; which replica $c_i^{(k)}$ is chosen is subject to optimization, and we will denote by μ the assignment of nodes to chunks.

In order to ensure a predictable application performance, both the connection to the chunks as well as the interconnection between the nodes may have to ensure certain minimal bandwidth guarantees; we will refer to the first type of virtual network as the *(chunk) access network*, and to the second type of virtual network as the *(node) inter-connect*; the latter is modeled as a complete network (a *clique*). Concretely, we assume that an active chunk is connected to its node at a minimal (guaranteed) bandwidth b_1, and a node is connected to any other node at minimal (guaranteed) bandwidth b_2.

C. Optimization Objective

Our goal is to develop algorithms which accept and embed a request whenever this is possible, and minimize the *resource footprint*: the amount of resources which have to be dedicated to a request, in order to realize its guarantees.

Formally, let $dist(v, c)$ denote the distance (in the underlying physical network T) between a node v and its assigned (active) chunk replica c, and let $dist(v_1, v_2)$ denote the distance between the two nodes v_1 and v_2. We define the *footprint* $\mathrm{F}(v)$ of a node v as follows:

$$\mathrm{F}(v) = \sum_{c \in \mu(v)} b_1 \cdot dist(v, c) + \underbrace{\frac{1}{2} \cdot \sum_{v' \in V \setminus \{v\}} b_2 \cdot dist(v, v')}_{\text{only for inter-connect}},$$

where $\mu(v)$ is the set of chunks assigned to v. Our goal is to minimize the overall footprint $\mathrm{F} = \sum_{v \in V} \mathrm{F}(v)$.

D. Problem Decomposition

In order to chart the landscape of the computational tractability and intractability of different problem variants, we decompose our problem into its fundamental aspects, namely replica selection (RS), multiple chunk assignment (MA), flexible node placement (FP), node interconnect (NI), and bandwidth constraints (BW), as described in the following. In this paper, we will consider all possible 32 problem variants, where each of these five aspects can either be enabled or disabled.

***Replica Selection* (RS).** The first fundamental problem is replica selection: if the input data is stored redundantly, the algorithm has the freedom to choose a replica for each chunk type, and assign it to a virtual machine (i.e., *node*). In the following, we will refer to a scenario with redundant chunks by RS; in the RS-only scenario, the number of chunk types is equal to the number of nodes. Otherwise, we will add the +MA property discussed next.

***Multiple Assignment* (MA).** If the number of chunk types τ is larger than the number of nodes, each node needs to be assigned multiple chunks. We will refer to such a scenario by MA. Since all nodes are identical and no additional information regarding the chunks is available at request time, we assume that each node will process an identical integer number of chunks $m = \tau / n_V$.

***Flexible Placement* (FP).** While the nodes are placed a priori in some cases, the node placement (or synonymously: *embedding*) of nodes on physical servers can also be subject to optimization. We will refer to this degree of freedom by FP.

***Node Interconnect* (NI).** We distinguish between scenarios where bandwidth needs to be reserved both from each node to its assigned chunks as well as to the other nodes (i.e., $b_1 > 0$ and $b_2 > 0$), and scenarios where only the (chunk) access network requires bandwidth reservation (i.e., $b_1 > 0$ and $b_2 = 0$). We will refer to the former scenario where bandwidth needs to be reserved also for the inter-connect, by NI. The node interconnect is modelled as a complete graph, to account for the all to all communication patterns of batch processing applications such as MapReduce.

***Bandwidth Capacities* (BW).** We distinguish between an uncapacitated and a capacitated scenario where the links of the substrate network come with bandwidth constraints, and will refer to the bandwidth-constrained version by BW; the capacity of servers (the number of nodes which can be hosted concurrently) is always limited. Note that capacity constraints introduce infeasible problem instances, where it is impossible to allocate sufficient resources to satisfy an embedding request.

III. Polynomial-Time Algorithms

Despite the various degrees of freedom in terms of embedding and replica selection, we can solve many problem variants efficiently. This section introduces three general techniques, which can roughly be categorized into *flow* (Section III-A), *matching* (Section III-B) and *dynamic programming* (Section III-C) approaches. First, let us make a simplifying observation:

Observation 1. *In problems without flexible placement (*FP*), the bandwidth required for the inter-connect network (*NI*) can be allocated* upfront*, as it does not depend on the replica selection and assignment. Accordingly, we can reduce problem variant* RS + MA + NI + BW *(as well as all its subproblems) to* RS + MA + BW *(resp. its subproblems).*

A. Flow Algorithms (RS + MA + NI + BW)

We first present an algorithm to solve the RS + MA + NI + BW problem. Recall that in this problem variant, we are given a set of redundant chunks (RS) and a set of nodes (the *nodes*) at fixed locations (no FP). The number of chunk types is larger than the number of nodes (MA), and each node needs to be connected to its selected chunks as well as to other nodes (NI), while respecting capacity constraints (BW). Our goal is to minimize the resource footprint F, consisting of the bandwidth reservations in the (chunk) access network and the (node) inter-connect. As we will see in the following, we can use a flow approach to solve this problem variant.

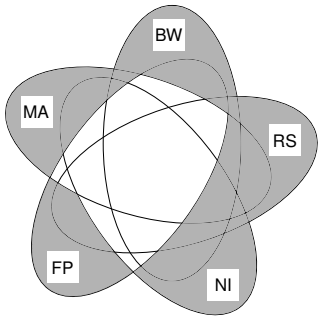

Fig. 2. Variants solved by flow approach.

Construction of Artificial Graph. In order to solve the RS + MA + NI + BW problem, we first remove the NI property using Observation 1. We then construct an artificial graph T^*, extending the substrate network T and normalizing bandwidth capacities, as follows. For T^*, we normalize the bandwidth of T to integer multiples of b_1, i.e., for each link $e \in E(T)$, we set its new capacity in T^* to $\lfloor cap(e)/b_1 \rfloor$. After this normalization, we extend the topology T by introducing an artificial vertex for each chunk type. These artificial vertices are connected to each leaf (i.e., server) in T where a replica of the respective chunk type is located, connecting the replica of the respective chunk type by a link of capacity 1. In addition, we create a *super-source* s^+, and connect it to each of the artificial chunk type vertices (with a link of capacity 1). Moreover, we create an artificial *super-sink* s^- and connect it to every leaf containing at least one node; the link capacity represents the number of nodes x hosted on this server, times the multi-assignment factor m. We additionally assign the following costs to edges of T^*: every edge of the original substrate network costs one unit, and all other artificial edges cost nothing.

A solution to the RS + MA + BW problem can now be computed a from a solution to the *Min-Cost-Max-Flow* problem between super-source s^+ and super-sink s^- on the artificial graph T^*.

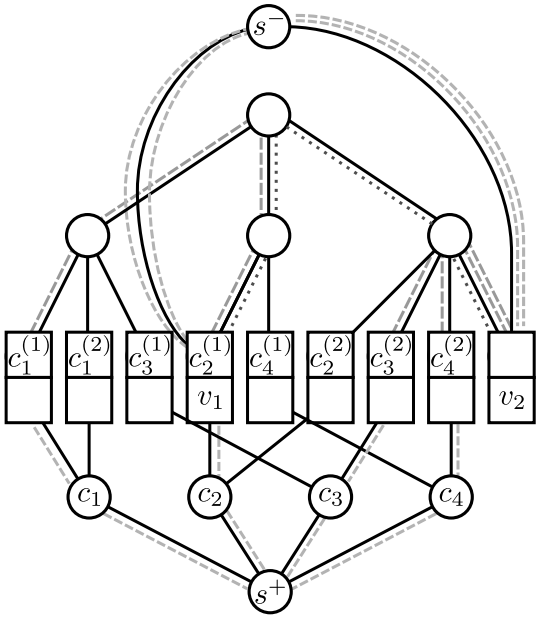

Fig. 3. Example of flow construction: Problem instance with two nodes, four chunk types, and two replicas per type. The min-cost-max-flow is indicated by the dashed lines: each line represents one unit of flow.

Example. Figure 3 shows an example of the extended substrate network T^*: The sink s^- is connected to the two leaves, which host the nodes. The artificial nodes are depicted below the leaves, are labeled with their respective chunk types (e.g., c_1), and are connected to the source s^+ as well as to the leaves which contain replicas of their chunk type. The maximum flow with minimal costs is indicated by the dashed lines: each line represents one unit of flow. The dotted lines indicate links which have reduced capacity due to NI.

Algorithm. Our algorithm to solve RS + MA + NI + BW consists of three parts: *First*, we construct the normalized and extended graph T^* described above and compute a min-cost-max-flow solution, e.g., using [18], [33]. *Second*, we have to *round* the resulting, possibly fractional flow, to integer values. Due to the *integrality theorem* [1], there always exists an optimal integer solution on graphs with integer capacities. However, while algorithms like the successive shortest path algorithm [24] directly give us such an integral solution (in polynomial time), the fastest min-cost-max-flow algorithms (e.g., based on double-scaling methods [18] or minimum mean-cost cycle algorithms [33], may yield fractional solutions which need to be rounded to integral solutions (of the same cost). In order to compute integral solutions, we proceed as follows: we iteratively pick an arbitrary (loop-free) path currently having a fractional allocation of value f ($f > 0$), and distribute its flow f among all other fractional paths of the same length; due to the optimality of the fractional solution and due to the integrality theorem, such paths must always exist. After distributing this flow, the total allocation on this path will be 0, and we have increased the number of integer paths by at least one. We proceed until we constructed the perfect matching. *Third*, given an integer min-cost-max-flow solution, we need to decompose the integer flow into the paths representing matched chunk-node pairs: The assignment can be obtained by decomposing the flow allocated in the original substrate network. In order to identify a matched chunk-node pair, we take an arbitrary (loop-free) path p carrying a flow of value ≥ 1 from s^+ to s^-: the first hop represents the chosen chunk type, the second hop the chosen replica, and the last but one hop represents the server: we will assign the replica to an arbitrary unused node on this server. Having found this pair, we reduce the flow along the path p by one unit. We continue the pairing process until every chunk type is assigned.

Analysis. The correctness of our approach follows from our construction of T^*, using integer capacities (in our case $\lfloor cap(e)/b_1 \rfloor$), and the fact that cost optimal integral solutions always exist [1]. The runtime of our algorithm consists of four parts: construction of T^*, computation of the min-cost-max-flow, flow rounding, and decomposition. The dominant term in the asymptotic runtime is the flow computation. Using the state-of-the-art min-cost-max-flow algorithms [18], [33] we get a runtime of $\mathcal{O}(n_S^2 \cdot \log\log\min\{U, \tau\})$ where U is the maximal link capacity; note that in networks with high capacity and uncapacitated networks, we can simply set $U = \tau$.

B. Matching Algorithms (RS+MA+NI *and* MA+NI+BW)

This section presents faster algorithms to solve the two problem variants RS + MA + NI and MA + NI + BW which can also be solved withthe flow approach introduced above. In general, we refer to the algorithms presented in this section as matching approaches.

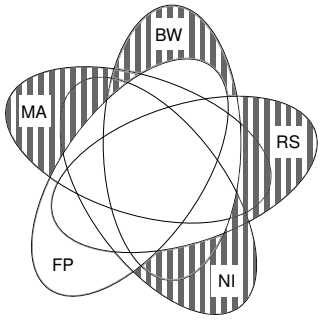

Fig. 4. Variants solved by matching approaches.

1) RS + MA + NI*:* Let us first consider the RS + MA + NI variant. Recall that in this problem, we are given a set of redundant chunks (RS) and a set of nodes at fixed locations. The number of chunk types is larger than the number of nodes (MA), and each node needs to be connected to its chunks as well as to other nodes (NI). Our goal is to minimize the resource footprint F, consisting of the bandwidth reservations in the access network and the inter-connect.

Algorithm. Due to Observation 1, RS + MA + NI degenerates to RS + MA. In order to solve the RS + MA problem variant, we construct a bipartite graph between the set V of nodes and the set of chunks. Concretely, we clone each node m times, as each node needs to process m chunk types, and we collect all copies of a given chunk type in a single "supernode". We connect each node to all chunk types using the *lowest hop count* to one of the copies as the cost metric (the link weight). On the resulting bipartite graph, we can now compute a *Minimum Weight Perfect Matching* [16]: the resulting matching describes the optimal assignment of chunks to nodes.

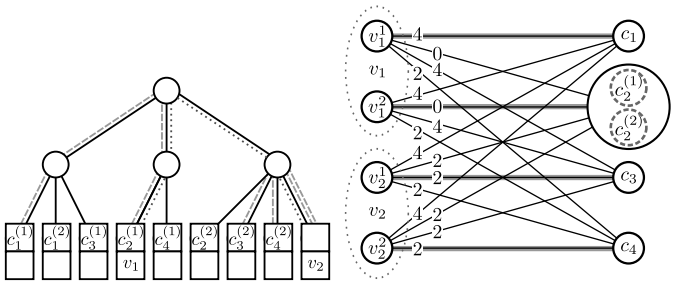

Fig. 5. The RS + MA problem on the *left* is converted into a matching problem on the *right*. Since each node has to process two chunks, the nodes are replicated in the matching representation. The two replicas of each chunk type are represented by a single node, and all edges connecting to this node have a weight according to the shorter distance to one of the replicas. This is visualized for c_2.

Example. Before analyzing our algorithm, let us consider a small example. Figure 5 illustrates an instance where two nodes are cloned into $m = 2$ nodes each, resulting in a total of four nodes in the matching problem representation. The two replicas of each chunk type are aggregated into a single chunk type vertex c_j in the matching problem; this gives a total of four chunk type vertices in the matching graph. The costs on the links between all clones of a specific vertex and a chunk type are set to the minimum distance. We can observe this for instance at the edges connecting the two clones of v_1 to c_2: both weights are 0.

Analysis. The correctness of our algorithm follows from the construction and the optimal solution of the minimum matching. The runtime consists of two parts: the construction of the matching graph and the actual matching computation. The constructed graph consists of $m \cdot n_V \cdot \tau$ many edges, and for each edge we need to compute its cost, i.e., the shortest distance which in a tree we can computed in time n_S; thus, the overall construction time is $\mathcal{O}(n_S \cdot \tau^2)$. The state of the art algorithm to compute matchings are based on scaling techniques [14]. The runtime translates to $\mathcal{O}(\tau^{5/2} \cdot \log(\tau \cdot n_S))$; recall that $\tau = m \cdot n_V$.

2) Faster MA + NI *and* MA + NI + BW*:* We now show that we can solve MA + NI even faster, by exploiting locality. Moreover, we will show that we can even solve MA + NI + BW problem variants by simply verifying feasibility. . In the following, due to Observation 1, we can focus on the MA resp. MA + BW problem.

We first introduce the following definition.

Definition 1 (Local Assignment (LA)). *We define an assignment μ to be* local in a specific subtree T'*, iff μ assigns the maximum number of chunks in the subtree to nodes in the same subtree. We define μ to be* local *when it is local with respect to all possible subtrees of the substrate network.*

Example. Figure 6 illustrates the concept of local assignment: The closest chunk to v_2 is c_1, and the closest node to c_1 is v_2. However, a subtree T' exists such that $v_1 \in T'$ and $c_1 \in T'$, but $v_2 \notin T'$. Therefore, a local assignment cannot assign c_1 to v_2.

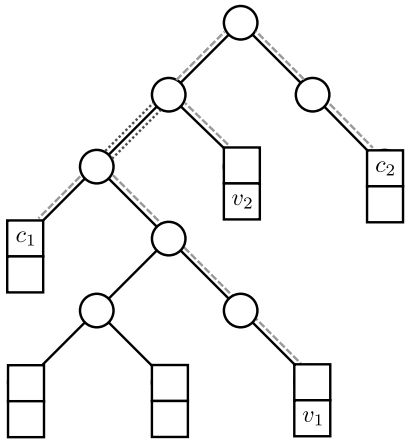

Fig. 6. Illustration of local assignment: The dashed lines indicate bandwidth allocations, which occur independently of the chosen assignment. The dotted lines indicate bandwidth allocation which occur only if c_2 is assigned to v_1.

We will see later that optimal solutions to MA have a local assignment. We exploit this in our algorithms described in the following.

Algorithm. Our proposed algorithm for MA proceeds in a bottom-up fashion, traversing the substrate network T from the leaves toward the root. For each subtree T', we maintain two sets S_1, S_2 in order to match unmatched chunks S_1 in the subtree T' to unmatched nodes S_2 in T'. Both sets are initially empty.

We first process all the leaves, in an arbitrary order; subsequently, we process arbitrary inner vertices of T, whenever all their children have been processed. We process any leaf ℓ by adding any nodes or chunks which are located on ℓ to the corresponding sets S_1 and S_2. A non-leaf vertex u is processed in the following way: we take the union of the sets of u's children, i.e., the sets contain the unmatched chunks and nodes in this subtree. For both leaves and inner nodes, whenever both sets are non-empty, we greedily match an arbitrary chunk in S_1 with an arbitrary node in S_2, and remove them from the sets.

Analysis. On a given vertex u, emptying one of the sets, results in a *local assignment* (cf Definition 1) in the subtree rooted at u. The bottom-up strategy ensures that this works for every subtree in the substrate, rendering the resulting assignment local. The complexity of this construction is low: For each vertex in the substrate graph, we build the union of the children's sets, and since each vertex can only be the child of one vertex, the amortized runtime per vertex is constant; and hence the overall runtime $\mathcal{O}(n_S)$. The sum of all remove operations, is equal to the number of chunk types $\mathcal{O}(\tau)$. Hence the overall complexity of this construction amounts to $\mathcal{O}(n_S + \tau)$.

It remains to prove optimality of such local assignments. We first characterize the bandwidth allocation on uplinks of subtrees.

Lemma 1. *Given an* MA *problem and a subtree T' containing x chunks and y nodes, the minimal bandwidth allocation*

of any assignment μ on the uplink of T' is $|x - y \cdot m| \cdot b_1$.

Proof. In case the number of chunk types equals the processing capacities of the nodes in the given subtree, the bandwidth allocation inflicted by the chunk access network on the uplink can be zero, since we can assign all chunks to nodes in the same subtree. Otherwise, we distinguish between two cases: Recall, that in instances without RS, all chunks have to be processed. In case there are more chunks in the subtree, at least all of the excess chunks have to be transferred to a different subtree, which will inflict costs b_1 per excess chunk on the uplink connecting T' with the remaining parts of T, which will inflict costs b_1 per excess chunk on the uplink of root of T'. Similarly, if the processing capabilities exceed the amount of available chunks, excess chunks from other subtrees will have to be transferred to nodes in the subtree T', inflicting bandwidth costs of b_1 each. Hence, the minimum bandwidth allocation for the chunk access on the uplink is the difference between the number of chunks and the processing capabilities of the subtree $|x - y \cdot m|$ times the amount of bandwidth needed, for a single transfer b_1. □

Theorem 2. *Given an* MA + NI *problem instance, a feasible assignment μ is optimal iff it is local.*

Proof. Local assignments generate exactly the minimal allocations on all links, as the assignments which generate the minimal bandwidth allocations described in the proof of Lemma 1 are local in the given subtree. Hence each local assignment has to be optimal. A non-local assignment, has at least one subtree, in which it is not local. This subtree will have a higher allocation on the uplink. Since the local assignment has minimal allocations on all other links, the non local assignment has a larger footprint. □

Combined with a simple postprocessing step, this approach can also solve MA+BW. The central idea of this extension, is that *local* assignments allocate the minimal bandwidth on each individual edge. In consequence, each bandwidth constraint which is lower than the allocation of a local assignment on one link, renders the problem infeasible. Hence, it is sufficient to temporarily omit the bandwidth limitations, compute an optimal assignment for an MA instance, and verify that the resulting allocations do not violate any capacities. The postprocessing step scales linearly with the number of edges in the substrate graph.

C. Dynamic Programming (MA + FP + NI + BW)

We now show how to solve the MA + FP + NI + BW problem variant in polynomial time. Note that this problem variant requires to find a tradeoff between the desire to place nodes as close as possible to each other (in order to minimize communication costs), and the desire to place nodes as close as possible to the chunk locations.

Example. Figure 8 shows an example: one extreme solution is to minimize the distance between chunks and nodes, see mapping π_1 in Figure 8 *(left)*: the four nodes are all collocated with chunks, resulting in a zero-cost chunk access network. As a result, the paths between the individual nodes are longer than in alternative node placements: each node has a distance of two hops to one other node, and four hops to two other nodes. Hence the resulting allocations for the node interconnect sum up to $20 \cdot b_2$. Figure 8 *(right)* shows a different node mapping π_2, which seeks to minimize the communication costs between the nodes, and places all nodes in one subtree. The distance between all nodes is two, which results in a total bandwidth allocation of $12 \cdot b_2$ for the interconnect. However, this reduced price comes at additional costs in the access network: c_3 and c_4 have to be communicated to v_3 and v_4, which requires a total bandwidth allocation of $8 \cdot b_1$.

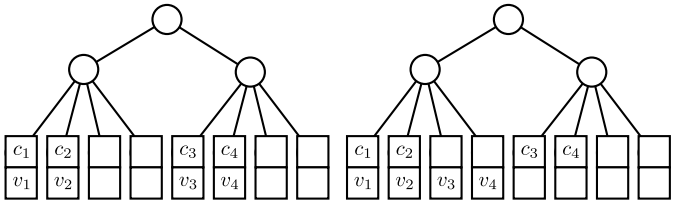

Fig. 8. Two different node placements for the same substrate graph and chunk locations. For $b_1 = b_2$, both solutions have an identical footprint. In other cases, one solution outperforms the other.

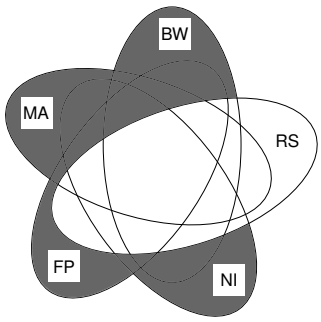

Fig. 7. Variants solved by dynamic programming approach.

Basic ideas. Our proposed approach is based on dynamic programming, and leverages the *optimal substructure property* of MA + FP + NI + BW: as we will see, optimal solutions for subproblems (namely subtrees) can efficiently be combined into optimal solutions for larger problems. Indeed, the MA + FP + NI + BW problem exhibits such a structure, and we show how to exploit it to compute efficient embeddings, even in scenarios where multiple chunks need to be assigned to flexibly placeable nodes.

For ease of presentation we will transform the substrate network T into a binary tree, using binarization: we clone every higher-degree node, iteratively attaching additional clones as right children and original children as left descendants.

As usual in dynamic programs, we define, over the structure of the tree, a recursive formula formula f for the minimal cost solution *given* any possible number of nodes embedded in a given subtree. The actual set does not matter, due to symmetry arguments. Our approach is to evaluate this function in a bottom-up manner. To finally compute the actual optimal embedding, we traverse the computed minimal-cost path backwards (according to the optimal values found for f during the bottom-up computation).

Concretely, the first argument to function f is a subtree T', containing a given number of chunks $y(T')$, and the

second argument is the number of nodes to be embedded in the subtree. Function f is evaluated in a bottom up manner. We initialize the function at each leaf ℓ, by $f(T_\ell, x) = \infty$ for all numbers of nodes x which are larger than the server capacity *cap*(ℓ); to calculate $f(T_\ell, x)$, for $x \leq$ *cap*(ℓ), we compute the bandwidth allocation on the uplink of T_ℓ, referred to by the function $bw(T_\ell, x)$: $bw(T_l, x) = b_1 \cdot |x - y(T_\ell)| + b_2 \cdot (n_V - x) \cdot x$, which accounts for the bandwidth allocation on the uplink of T_ℓ. The first term represents the required bandwidth for the communication between the x nodes on ℓ, and the $n_V - x$ nodes in the remaining parts of the substrate network. The second term represents the bandwidth, which is necessary to transport the chunks from their location to the node which should process the data (see Lemma 1 for more details).

After initialization, we proceed to compute f for non-leaf nodes in a bottom-up manner: We split the x nodes into two positive integer values, and we put r on the right and $x - r$ on the left subtree. That is, we take the optimal cost (given recursively) of placing r nodes in the right subtree $\text{RI}(T')$ of T' and $x - r$ nodes in left subtree $\text{LE}(T')$ of T'. Given the cheapest combination, we add the bandwidth requirements on the uplink of T' to generate the overall costs for placing x nodes in T'.

$$f(T', x) = \min_{0 \leq r \leq x} \{f(\text{LE}(T'), x - r) + f(\text{RI}(T'), r)\} + bw(T', x)$$

Again, we set $f(T', x)$ to infinity if the required bandwidth bw exceeds the capacity *cap* of the uplink of T'.

Analysis. The correctness and optimality of our dynamic program is due to the decoupling of the costs induced by the tree structure of T and the substructure optimality property. The substructure optimality follows from the observation that costs can be accounted on the uplink, and the fact that we check each possible node distribution. For each substrate vertex (n_S many) we have to check the cost of all possible splits, resulting in an overall complexity of $\mathcal{O}(n_S \cdot n_V^2)$. The runtime to binarize T is asymptotically negligible.

Remark: Simple Problems. To conclude, for the sake of completeness, we also observe that there are several problems which allow for a trivial solution. Concretely, problems with FP plus any combination of RS and BW (but without MA and NI) can easily be solved by mapping nodes to chunk locations.

IV. NP-Hardness Results

We have seen that even problems with multiple dimensions of flexibility can be solved optimally in polynomial time. This section now points out fundamental limitations in terms of computational tractability. In particular, we will show that problems become NP-hard if flexibly placeable nodes (FP) have to be assigned to one of multiple replicas (RS), either with multiple chunks per node (MA in Section IV-B) or with communication among nodes (NI in Section IV-C). both results hold even in uncapacitated networks, and even in small-diameter substrate networks (namely two- or three-level trees [2]). The hardness of FP + RS + MA and FP + RS + NI imply the hardness of four additional, more general models, as summarized in Figure 9:

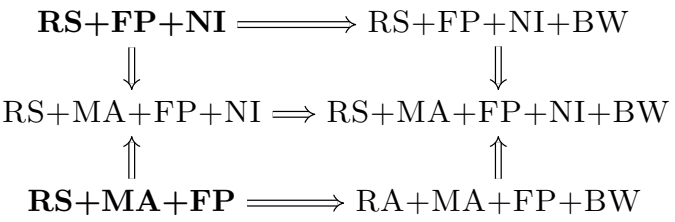

Fig. 9. The NP-hardness of 2 variants, implies that 4 other variants are also NP-hard.

A. Introduction to 3D Perfect Matching

Both the hardness of FP + RS + MA and FP + RS + NI are shown by a reduction from the NP-complete problem of *3D Perfect Matching* [10], which we can see as a generalization of bipartite matchings to 3-uniform hypergraphs. We will refer to this problem by 3-DM, and for completeness, review it quickly: 3-DM is defined as follows. We are given three finite and disjoint sets X, Y, and Z of cardinality k, as well as a subset of triples $T \subset X \times Y \times Z$. Set $M \subseteq T$ is a 3-dimensional matching if and only if, for any two distinct triples $t_1 = (x_1, y_1, z_1) \in M$ and $t_2 = (x_2, y_2, z_2) \in M$, it holds that $x_1 \neq x_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$. Our goal is to decide if we can construct a $M \subseteq T$ which is *perfect*, that is, a subset which covers all elements of $X \times Y \times Z$ exactly once.

B. Multi-Assignments are hard (FP + RS + MA)

Our proof that FP + RS + MA is NP-hard is based on the following main ideas. We encode a 3-DM instance as an FP + RS + MA instance as follows:

- For every element in the universe $X \cup Y \cup Z$, we create a chunk type. Intuitively, in 3-DM, each element must be covered, which corresponds to the requirement of FP + RS + MA that each chunk type is processed.
- We will encode each triple as gadget with three leaves in a substrate tree T. The three leaves are close to each other in T, and the placement of chunk replicas in FP + RS + MA corresponds to the elements of the triples in these leaves.
- The node placement will correspond to the choice of triples, independently of which leaf the node is mapped to. A node will process its collocated chunk, as well as the chunks in other two leaves of the same gadget.
- In order to turn the optimization problem into a decision problem, we will use a cost threshold Th. The cost threshold will be met by all assignments which assign all three chunks of each triple to a node which is collocated with one of the chunks. Assignments which connect a chunk to a node in a different triple, will have a larger footprint, and are considered to be infeasible.

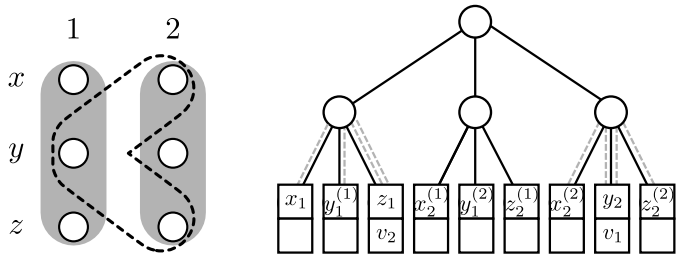

Fig. 10. *Left:* A 3-DM instance with three triples: (x_1, y_1, z_1), (x_2, y_1, z_2), and (x_2, y_2, z_2). The solution is indicated by the grey triples; the dashed triple is not used for the solution. *Right:* The corresponding problem and solution of FP + MA + RS.

Construction. Given an instance I of 3-DM in which k triples have to be choosen, we construct an instance I' of FP + RS + MA as follows:

- *Tree Construction:* We create a tree consisting of a root, and for each triple, we create a gadget which we directly attach as child of the root. The gadget is of height 2, and has the following form: The gadget of each triple consists of an inner node (a router) and three leaves.
- *Chunks and chunk replicas:* For each element in X, Y and Z, we create a chunk type ($3 \cdot k$ in total). Every gadget contains three chunk replicas, corresponding to the elements of the triple. Each leave in a gadget, contains exactly one replica.
- *Other properties:* We set the number of to-be-embedded nodes to k, b_1 to 1, and the number of chunk slots in each node to the multi-assignment factor $m = 3$. We use a threshold $Th = 4 \cdot k$.

Example. Figure 10 shows an example of our construction: An instance I of 3-DM is given: The disjoint sets X, Y and Z have a cardinality $k = 2$. We will refer to the two elements in X as x_1 and x_2, and use the same notation for the other two sets. T contains the three triples (x_1, y_1, z_1), (x_2, y_1, z_2), and (x_2, y_2, z_2). The goal of 3-DM is to find a subset $M \subseteq T$, which contains each element in each of the three sets exactly once. This instance only has one solution: $M = \{(x_1, y_1, z_1), (x_2, y_2, z_2))\}$.

To construct the corresponding instance I' of FP + RS + MA, we create a gadget for each triple in T. For each variable which occurs in a triple, the corresponding gadget contains a chunk of the type of the variable. The triple (x_2, y_1, z_2) of the instance is represented by the middle gadget in Figure 10. The objective of I' is to spawn $k = 2$ nodes, with the smallest possible footprint. If the total footprint is $\leq 2 \cdot 2 \cdot k$, we can construct a solution to I from the solution to I'. The footprint consists of the costs which occur when a node is embedded in a gadget, and the three chunks of that gadget which are assigned to that node: one of the chunks is collocated with the node, the other two have to be transferred via two hops, inflicting unitary costs on each hop.

Correctness. Given these concepts, we can now show the computational hardness.

Theorem 3. FP + RS + MA *is NP-hard.*

Proof. Let I be an instance of 3-DM and let I' be an instance of FP + RS + MA constructed as described above. We prove that I' has a solution of cost $\leq Th$ if ($\Rightarrow$) and only if ($\Leftarrow$) I has a matching of size k.

($\Rightarrow$) Let us take a solution to 3-DM. We place a node in every gadget that corresponds to the chosen triples. In each of the corresponding gadgets, we match every chunk to the node in this gadget. This solution has cost exactly Th. As every element of the universe is covered, every chunk type is processed.

($\Leftarrow$) Let us take a solution to FP + RS + MA of cost $\leq Th$. We choose triples that correspond to gadgets where there are nodes. Since all chunks are processed, every element of X, Y and Z is matched. Each node must process chunks that correspond to the triple, otherwise the cost must be larger than Th (high costs for chunk transportation). □

C. Inter-connects are hard (FP + RS + NI)

Next, we prove that the joint optimization of node placement and replica selection is NP-hard if an inter-connect has to be established between nodes. In our terminology, this is the FP + RS + NI problem.

The proof is similar in spirit to the proof of FP + RS + MA, however, we modify the construction to account for the absence of MA: we choose a high value for b_1, such that nodes will be directly collocated with their assigned chunks. We leverage the fact that any solution which does not assign 0 or 3 chunks to each gadget, will have higher communication costs.

Construction. Let I be an instance of 3-DM. We will create an instance I' for FP + RS + NI as follows:

- We will construct the same tree as in previous reduction with chunk replicas placed in the same way.
- The communication cost in the inter-connect is set to $b_2 = 1$.
- The number of nodes (virtual machines) is $n_V = 3 \cdot k$, where k is the set cardinality.
- Only solutions which place a node in each leaf of k gadgets, can be converted into solutions for the 3-DM problem. We use the cost threshold $Th = 6 \cdot k + 18 \cdot (k-1) \cdot k$, to verify whether a solution achieves this, transforming FP + RS + NI into a decision problem. A detailed explanation of this value can be found in the proof of Theorem 5.
- We set the access cost b_1 to a chunk replica to a high value W. This will force nodes to be collocated with the replica. One example of sufficient (and polynomial but not necessarily minimal) W is the value of the threshold $Th + 1$. Any solution not assigning chunks to collocated nodes, have cost $> Th$: communicating a chunk inflicts costs $W = Th + 1$ over every link.

We focus on instances with unit server capacities.

Proof of correctness. Intuitively, in order to minimize embedding costs, nodes should be placed on near-by replicas. We use the following helper lemma.

Lemma 4. *In every valid solution of I' of cost $\leq Th$, each gadget falls in one of two categories: k gadgets have exactly 3 nodes, and $n-k$ gadgets remain empty.*

Proof. Since W is large enough, the $3 \cdot k$ nodes have to be placed directly on different chunks, resulting in 0 costs for the access network. Consider any pair of nodes communicating over the inter-connect; due to our construction, the communication cost for each such pair is either 2 hops (if they belong to the same gadget) or 4 hops (if they belong to different gadgets). The lemma then follows from the observation that Th is chosen such that it is never possible to distribute nodes among more than k gadgets. □

Theorem 5. FP + RS + NI *is NP-hard.*

Proof. Let I be an instance of 3-DM and let I' be an instance of FP + RS + NI constructed as described above. We prove that I' has solution of cost $\leq Th$ if ($\Rightarrow$) and only if ($\Leftarrow$) I has a solution.

($\Rightarrow$) In order to compute a solution for I' given a solution for I, we proceed as follows. Given an exact covering set of triples $S = \{t_1, t_2, \ldots, t_k\}$, we place three nodes in each gadget that corresponds to every triple of S. Chunks are matched to the nodes which are located on the same server.

The solution has the following cost: (1) the communication cost inside a gadget is $2 \cdot \binom{3}{2}$, as every pair contributes two hops; (2) the communication cost from each gadget to all other gadgets is $4 \cdot 3 \cdot 3 \cdot (k-1)/2$, where the factor 4 is for the communication over 4 hops, the factor 3 corresponds to the number of nodes per gadget, and $3 \cdot (k-1)$ is the number of nodes in remote gadgets; as we count each pair twice, we need to divide by two in the end. Summing up over all k gadgets, we get exactly Th.

($\Leftarrow$) Given a solution for I', we can exploit Lemma 4 to construct a solution for I. We know that in any solution of cost at most Th, k gadgets contain exactly 3 nodes. These gadgets correspond to a valid 3D Perfect Matching: exactly one replica of every chunk type is processed and hence every element is covered exactly once. □

V. Related Work

There has recently been much interest in programming models and distributed system architectures for the processing and analysis of big data (e.g. [3], [11], [35]). The model studied in this paper is motivated by MapReduce [11] like batch-processing applications, also known from the popular open-source implementation *Apache Hadoop*. These applications generate large amounts of network traffic [8], [25], [37], and over the last years, several systems have been proposed which provide a provable network performance, also in shared cloud environments, by supporting relative [26], [27], [31] or, as in the case of our paper, *absolute* [5], [21], [28], [29], [34] bandwidth reservations between the virtual machines.

The most popular virtual network abstraction for batch-processing applications today is the *virtual cluster*, introduced in the Oktopus paper [5], and later studied by many others [25], [30], [34]. Several heuristics have been developed to compute "good" embeddings of virtual clusters: embeddings with small footprints (minimal bandwidth reservation costs) [5], [25], [30], [34]. The virtual network embedding problem has also been studied for more general graph abstractions (e.g., motivated by wide-area networks). [9], [15]

From a theoretical perspective, the virtual network embedding problem can be seen as a generalization of classic VPN graph embedding problems [19], [22], in the sense that in virtual network embedding problems, also the embedding endpoints are flexible. In this sense, the virtual network embedding problem can also be seen as a generalization of the classic NP-hard Minimum Linear Arrangement problem which asks for the embedding of guest graphs on a simple *line topology* (rather than tree-like topologies as studied in this paper) [12], [13].

However, to the best of our knowledge, we are the first to provide an algorithmic study of the virtual cluster embedding problem which takes into account data locality as well as the possibility to select replicas—aspects which so far have only been studied from a best-effort perspective and using coarse-grained metrics (e.g., same rack or same server), thus limiting the flexibility of the system [4], [23], [36].

VI. Summary and Conclusion

At the heart of locality and replica aware virtual cluster embeddings lie fundamental algorithmic problems. This paper has shown that despite the multiple dimensions of flexibility in terms of chunk assignment and node placement, many problems can be solved in polynomial time. However, we have also shown that several embedding problems are NP-hard already in two- and three-level trees—a practically relevant result given today's datacenter topologies [2])-.

Our results are summarized in Table I. One interesting take-away from this figure regards the question which properties render the problem NP-hard. For instance, we see that, BW does not influence the hardness of any problem variant, while RS is crucial for NP-hardness. MA only affects hardness if combined with RS. NI is trivial without FP, and FP requires more sophisticated algorithms when combined with NI or MA; in combination with RS and MA or NI, FP renders the problem NP-hard.

Acknowledgments. We would like to thank our shepherd Kihong Park. Research in part supported by Polish National Science Centre grant DEC-2013/09/B/ST6/01538.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, pages 63–74, 2008.

[3] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. Nodb: Efficient query execution on raw data files. In *Proc. ACM SIGMOD*, pages 241–252, 2012.

[4] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: Coping with skewed content popularity in mapreduce clusters. In *Proc. EuroSys*, pages 287–300, 2011.

[5] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proc. ACM SIGCOMM*, 2011.

NP-hard	5 combinations	RS + MA + FP + NI + BW
	4 combinations	RS + MA + FP + NI; RS + MA + FP + BW; RS + FP + NI + BW
	3 combinations	RS + MA + FP; RS + FP + NI
Flow	4 combinations	RS + MA + NI + BW
	3 combinations	RS + NI + BW; RS + MA + BW
	2 combinations	RS + BW
DP	4 combinations	MA + FP + NI + BW
	3 combinations	MA + FP + NI; MA + FP + BW; FP + NI + BW
	2 combinations	MA + FP; FP + NI;
Matching	3 combinations	RS + MA + NI; MA + NI + BW
	2 combinations	RS + MA; RS + NI; MA + NI; MA + BW; NI + BW
	1 combinations	RS; MA; NI; BW
0 Cost	3 combinations	RS + FP + BW
	2 combinations	RS + FP; FP + BW
	1 combinations	FP

TABLE I
FASTEST ALGORITHMS FOR DIFFERENT RESPECTIVE PROBLEM VARIANTS.

[6] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow.*, 1(2), 2008.

[7] M. Chowdhury, S. Kandula, and I. Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *Proc. SIGCOMM*, pages 231–242, 2013.

[8] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing Data Transfers in Computer Clusters with Orchestra. In *Proc. ACM SIGCOMM*, 2011.

[9] N. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Comput. Netw.*, 54(5):862–876, 2010.

[10] P. Crescenzi, V. Kann, M. Halldorsson, M. Karpinski, and G. Woeginger. Maximum 3-dimensional matching. *A Compendium of NP Optimization Problems*, 2000.

[11] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. USENIX OSDI*, pages 137–150, 2004.

[12] N. R. Devanur, S. A. Khot, R. Saket, and N. K. Vishnoi. Integrality gaps for sparsest cut and minimum linear arrangement problems. In *Proc. ACM STOC*, 2006.

[13] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[14] R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1413–1424, 2012.

[15] A. Fischer, J. Botero, M. Beck, H. DeMeer, and X. Hesselbach. Virtual network embedding: A survey. 2013.

[16] H. Gabow. A scaling algorithm for weighted matching on general graphs. In *Proc. IEEE FOCS*, 1985.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proc. ACM SOSP*, pages 29–43, 2003.

[18] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989.

[19] N. Goyal, N. Olver, and F. B. Shepherd. The VPN conjecture is true. *Proc. ACM STOC*, 2008.

[20] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM*, 2009.

[21] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *Proc. 6th CoNEXT*, 2010.

[22] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network. In *Proc. ACM STOC*, New York, New York, USA, 2001.

[23] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proc. ACM SOSP*, pages 261–276, 2009.

[24] Z. Kiraly and P. Kovacs. Efficient implementations of minimum-cost flow algorithms. In *ArXiv Technical Report 1207.6381*, 2012.

[25] J. C. Mogul and L. Popa. What we talk about when we talk about cloud network performance. *ACM SIGCOMM CCR*, 2012.

[26] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: Sharing the network in cloud computing. In *Proc. HotNets-X*, 2011.

[27] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. In *Proc. ACM SIGCOMM*, pages 351–362, 2013.

[28] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. Cloud control with distributed rate limiting. In *Proc. SIGCOMM*, 2007.

[29] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proc. 3rd Conference on I/O Virtualization (WIOV)*, 2011.

[30] M. Rost, C. Fuerst, and S. Schmid. Beyond the stars: Revisiting virtual cluster embeddings. In *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, 2015.

[31] A. Shieh, S. Kandula, A. Greenberg, and C. Kim. Seawall: Performance isolation for cloud datacenter networks. In *Proc. USENIX HotCloud*, 2010.

[32] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[33] E. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, July 1985.

[34] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: incorporating time-varying network reservations in data centers. *ACM SIGCOMM Computer Communication Review (CCR)*, 2012.

[35] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *Proce. ACM SIGMOD*, pages 13–24, 2013.

[36] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. EuroSys*, pages 265–278, 2010.

[37] Measuring EC2 system performance. http://goo.gl/V5zhEd.